DYNAMIC AND CONTINUOUS-TIME SERVICE NETWORK DESIGN

A Dissertation
Presented to
The Academic Faculty

By

Luke Marshall

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology

May 2018

DYNAMIC AND CONTINUOUS-TIME SERVICE NETWORK DESIGN

Approved by:

Dr. Natashia Boland (thesis advisor)
Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Martin W.P. Savelsbergh (thesis advisor)
Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Alan L. Erera
Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Chelsea C. White III
Industrial and Systems Engineering
Georgia Institute of Technology

Dr. John-Paul B. Clarke
Aerospace Engineering
Georgia Institute of Technology

Date Approved: December 14, 2017

Sometimes science is a lot more art than science

Rick Sanchez

ACKNOWLEDGEMENTS

The last three and a half years has been a life changing experience. There has been many challenges, both academic and personal, which has forced me to grow as a scholar, and human being – and there has also been many fun and exciting times with some truly wonderful people. All in all, I have loved my time at Georgia Tech.

First and foremost, I want to thank my advisors Natashia Boland, and Martin Savelsbergh for the opportunity to learn from their experience: I greatly admire your dedication and appreciate how available you have been to give useful and in-depth feedback. Thank you for the countless hours we have worked together, I certainly would not be here without you. I also want to thank my committee members: Alan Erera, Chip White, and J.P. Clarke; your suggestions and approval means a lot to me. In particular, I had the honor to collaborate with Alan in both a research and an administration setting (GSAC). In addition, my other collaborators, Mike Hewitt (from Loyola University Chicago), and Iman Dayarian – it has been an absolute pleasure working with all of you.

Special thanks to Dr Michael Forbes, who first introduced me to Operations Research in my B.Sc. His classes and mentorship inspired me to pursue this field and a PhD. I am incredibly grateful for the impact he has had on my life. I also want to thank the many wonderful faculty members at Georgia Tech – not only are they incredibly talented, they are also down-to-earth and encouraging. In particular, Profs. Tovey, Dey, Torriello, Andradottir, and Monteiro; I appreciate each of your styles, and have learnt many invaluable lessons from your perspectives.

Finally, but certainly not least, a big thank you to my friends and family. You all have made my time at Georgia Tech enjoyable, and have been an incredible support in the most challenging of times. In particular I want to thank Timur Tankayev and Mina Georgieva, I will always cherish our friendship. Also, Asteroide Santana and Seyma Güven-Koçak for our solidarity with the many weeks spent studying for the comprehensive exams. There

are so many wonderful and talented people in ISyE that I call friends – there are far too many names to list, but I thank you all for our time together. Special thanks to my amazing partner Joanna Whisnant; you have brought so much joy and love into my life. Lastly, I thank my family for their caring support, in particular my parents who raised a math geek coder, and encouraged him to chase his dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The thesis focuses on two fundamental problems in transportation and logistics, namely, service network design, and operation, with a focus on high precision, and large scale.

A typical approach to solving the design problem is by modeling with time-expanded networks and solving using integer programming, however this often yields an approximation to the continuous-time optimal solution. We investigate the price of this approximation caused by the discretization of parameters involving time, and introduce two algorithms that efficiently solve the continuous-time problem.

Both algorithms dynamically build and refine a subset of the full time-expanded network, so that the associated integer program is more computationally tractable, while still providing a guarantee of continuous-time optimality. While both approaches share the same overall structure, as well as many modeling similarities, the predominant difference is in the dynamic refinement step, and the underlying principles used to guarantee convergence.

The second algorithm is further extended to support in-tree loading, and freight splitting. In-tree loading simplifies operational overhead by requiring freight with common ultimate destination cross-docked at a terminal to travel along the same path; in this way terminal operators need only look at the ultimate destination in order to load shipments. Freight splitting allows for increased utilization by arbitrarily breaking shipments into smaller pieces; it is also a modeling technique to support aggregating shipments with common origin/destination in order to keep the model size tractable.

The design problem is primarily concerned with the routing of freight and service capacity, and is typically solved infrequently using predicted freight, whereas the operation problem is highly dynamic, using actual day-to-day volumes, and focuses on loading/dispatching vehicles, as well as crew and resource scheduling. We introduce an efficient heuristic to solve a large scale real-life operation problem, as well as providing new and useful metrics for evaluating operational performance.

CHAPTER 1
INTRODUCTION AND BACKGROUND

Our lives depend on efficient freight transportation. That may sound overly dramatic, but it's actually hard to overstate its importance, since it effects nearly every part of our life - for example: food, clothing, shelter, and health-care all rely on timely freight transportation. Moreover, it is also a fundamental component of the world economy, by supporting production, trade, and consumption (Crainic, [2000]). It has been shown to have a strong relationship, and even be a useful predictor of a country's economic growth (Brumbaugh et al., [2016]). As such, it is imperative that transportation problems are solved efficiently and with high precision. In particular, two fundamental problems are service network design, and operation: *design* typically involves the choice of routes (that freight follows from its origin to its destination), available service capacity, and resource repositioning (i.e. empty trailers); *operation* is highly dynamic and focuses on the loading and dispatching of vehicles, crew scheduling, and resource allocation (Crainic and Laporte, [1997]). This thesis will focus on both of these problems with the view of high precision, and large scale. The first part will focus on the quality of service design, while the second part will focus on service operation in an applied setting with a national US LTL carrier.

Given a network and set of commodities, the Continuous-Time Service Network Design Problem (CTSNDP) chooses an optimal route for each commodity, and consolidates them such that the total cost of transportation is minimized. A route for a commodity is a path through space and time, starting from its origin (after a given available time) and ending at its destination (before a given due time) with a dispatch time at each visited location along its path (i.e., the time when the commodity leaves the location). Many optimization problems that need to decide *when* an action occurs are often conveniently modeled using a time-expanded network. This approach discretizes the time horizon into intervals and maps

any parameters involving time to these intervals. Depending on the discretization scheme, the resulting model can yield an approximate, or a continuous-time optimal solution.

Time-expanded networks are popular since they provide a flexible modeling paradigm, which can be used with powerful commercial mixed integer programming (MIP) solvers for their solution. However, choosing an appropriate time discretization can be challenging. This choice directly impacts the size and quality of the associated MIP: *fine* discretization schemes typically give good approximations to the continuous-time problem at the expense of large, perhaps intractable, models; whereas *coarse* discretization schemes are more computationally amenable, but generally yield poor approximations. In practice, the choice of the time discretization is typically dominated by computational tractability (i.e., models that can fit into memory and be solved within an acceptable time frame), and as a consequence, little, if anything, is known about the loss of quality resulting from time discretization.

The primary goal of the second chapter is to assess and quantify this "price of discretization" for the service network design problem by means of an extensive computational study. The results will, of course, be specific to the service network design problem (and, too, to the specific instances used in the study), but we hope and expect that the findings are relevant also in other contexts where time-expanded network models are (or can be) used. The results show that the price of discretization can be high, with a loss of solution quality of more than 20% in some instances (see Figure 2.1). Even a small loss of solution quality can greatly impact the profitability of companies employing service network design models on a regular basis, such as parcel delivery companies like UPS and FedEx. Moreover, in recent years there has been an increased emphasis on customer service (timely delivery) - thus it is increasingly important to be aware of the loss of missed opportunities that are introduced by coarse grained approximations. We find that this loss of solution quality is primarily due to the loss of low-cost feasible paths, and loss of potential consolidations; and can be somewhat characterized by the *flexibility* of an instance (defined in Section 2.4.2).

Instances with high flexibility are less effected by coarse discretization, but can be more difficult to solve due to the increased consolidation opportunities.

Chapter 3 introduces an iterative algorithm for solving the Continuous-Time Service Network Design Problem (CTSNDP), which dynamically chooses and updates the discretization of a time-expanded network model in order to guarantee a continuous-time optimal solution, while being able to solve only relatively small mixed integer programs. Moreover, this algorithm iteratively provides feasible solutions with a guaranteed optimality gap, and so can be used as a heuristic with confidence in quality. We believe our Dynamic Discretization Discovery algorithm (DDD) can be adapted to other continuous-time problems that can be solved using time-expanded network models. Similarly in Chapter 4, we provide an alternate DDD algorithm for solving CTSNDP which focuses on connecting time intervals, instead of the first approach, which focuses on representing a set of travel times in a single arc. We then extend our investigation in Chapter 5, by allowing freight splitting, and in-tree loading, so that our research has broader practical relevance.

Next, in the second part of this dissertation, we look at service *operation* in large scale environments. In practice, coordinating daily operational tasks is highly dynamic. Where service *design* is a tactical in nature and focusing on routing freight, using predicted demands, and ensuring available service capacity, service *operation* handles/deals with varying day-to-day demands and scheduling resources (crew and equipment). Thus the service network is designed infrequently, say, every month; plans and schedules can then be built, say, a week in advance, to prepare for 'todays' activities. Then, on the day of operation, when the actual freight is known, these plans and schedules can be adjusted in order to reduce the cost of transport, or to meet service guarantees. Due to the highly dynamic daily environment, any plan and schedule adjustments need to be decided quickly. The scale and complex real-life constraints make this an enormous challenge. Chapter 6 introduces an efficient heuristic for making plan and schedule adjustments, that can handle

large-scale real-life instances, and has been successfully applied at a national US LTL carrier.

1.1 Contribution

To date, there has not been any quantitative study that thoroughly explores the price of discretizing time for service network design. Our computational study changes that situation (Chapter 2). The results, which took several months to compile, provide valuable insights into the price of discretizing time. Having a better, quantitative understanding of the (potentially negative) impact of the choice of time discretization is informative and useful, and may prompt researchers and practitioners to be more careful when choosing a time discretization, but it does not, in and of itself, provide any mechanism to assist with that choice. We address this issue in two ways.

Firstly, we examine whether there are problem characteristics or metrics that have predictive value which may help choose an appropriate time discretization (or at least prevent choosing a poor time discretization).

Secondly we introduce two algorithms that dynamically discover a suitable non-uniform discretization that gives bounds on the approximation, or can be solved to the continuous-time optimal solution (Chapters 3 & 4). These algorithms have additional computational benefits, and many instances can be solved orders of magnitude faster than when modeling using standard uniform discretization. We believe that both these algorithms can be applied to other similar problems that are modeled using use time-expanded networks.

We note that the algorithm described in Chapter 3 is joint work with Mike Hewitt from Department of Information Systems and Supply Chain Management, Quinlan School of Business, Loyola University Chicago. While my contributions to that research have been limited, because of the significant overlap of the research, we believe it suitable to include in this dissertation.

Finally, for the service operation problem, we develop an efficient heuristic algorithm which has been applied with success at a national US LTL carrier (Chapter 6). This algorithm has been designed to be run daily (possibly multiple times per day) on large scale instances, and to incorporate and mimic complex business logic. In the process, new and useful evaluation metrics have been introduced, and combined with clever data validation techniques, this has allowed us to deliver detailed reports on historical choices to compare with our suggested improvements. In addition, advanced schedule cancellation rules have been developed to reduce transportation costs, and our efficient freight rerouting techniques significantly increase on-time delivery. Furthermore, the framework has been designed to be easily extended, and so this approach can be beneficial to many other service operation applications.

Part I

Service Network Design

CHAPTER 2
THE PRICE OF DISCRETIZATION

2.1 Introduction

Freight transportation is arguably one of the most important components of the economy; supporting production, trade, and consumption (Crainic, [2000]). As such, it is critical to solve transportation problems efficiently and with high precision. In addition, the impact of time (deciding *when* activities occur), has become a critical consideration, due to the rise of same day delivery, just-in-time manufacturing, and competitive service level guarantees. The timing of activities include, for example, when a truck should leave a distribution center or when a crew should commence its duty. Many problems that need to make these *timed* decisions can naturally and conveniently be represented using a time-expanded network and solved using mixed integer programming (MIP).

Networks are extremely versatile, and can be used when modeling applications in transportation, manufacturing, communication, and scheduling. See Ahuja, Magnanti, and Orlin, (1993) for a more complete list. A time-expanded network is often constructed from a *static* (or *flat*) network by: partitioning the planning horizon into discrete intervals (often of equal length, known as *uniform*), duplicating the original network's nodes at each interval, and appropriately adding arcs to connect the, now timed, nodes. For example, a node in a time-expanded network could represent a location at a point in time, while an arc could represent the transportation of a container from one location to another, implicitly including the transit duration and time of dispatch. This approach is a common modeling technique, and was originally introduced by Ford and Fulkerson, (1958) to solve dynamic network flow problems.

Time-expanded networks are popular since they provide a flexible modeling paradigm,

and the associated integer programming formulations can be solved using powerful commercial software packages. However, choosing an appropriate time discretization can be challenging. This choice directly impacts the size of the associated integer program and the quality of its solution: *fine* discretization schemes typically give good approximations to the continuous-time problem at the expense of large, perhaps intractable, integer programs; whereas *coarse* discretization schemes are more computationally amenable, but generally yield poor approximations. In practice, the choice of the time discretization is typically dominated by computational tractability (i.e., integer programs that can fit into memory and can be solved within an acceptable time frame), and as a consequence, little, if anything, is known about the loss of quality resulting from time discretization.

The loss of quality is introduced by the adjustment of parameters that involve time. Because time is discretized, the parameters involving time, such as transit times and due times, need to be appropriately mapped to the discrete intervals, that is, a rounding scheme has to be employed. The three most intuitive schemes are *optimistic*, *regular*, and *conservative*, which are analogous to rounding operators floor, nearest, and ceiling (respectfully). Table 2.1 illustrates these schemes in a transportation setting for time window parameters (early and late times – e and l) and transit time parameters (τ).

Table 2.1: Rounding Schemes

Scheme	Early	Late	Transit
Optimistic	$\lfloor e \rfloor$	$\lceil l \rceil$	$\lfloor \tau \rfloor$
Regular	$\lfloor e \rceil$	$\lfloor l \rceil$	$\lfloor \tau \rceil$
Conservative	$\lceil e \rceil$	$\lfloor l \rfloor$	$\lceil \tau \rceil$

Conservative rounding schemes underestimate availability (e.g. round up travel times and shrink time windows); optimistic schemes do the opposite (e.g. round down travel times and enlarge time windows); and regular schemes simply round to the nearest discretized value. With conservative rounding, it is possible that certain continuous-time feasible solutions are no longer feasible in discretized time, leading to loss of solution

quality. Conversely, with optimistic rounding, it is possible that certain feasible solutions in discretized time are not continuous-time feasible, and hence cannot be used in practice. Because of this, we focus our efforts on conservative rounding schemes, since they have the desirable property that any feasible solution to the time-expanded network model can be implemented in continuous-time, albeit at a price to solution quality.

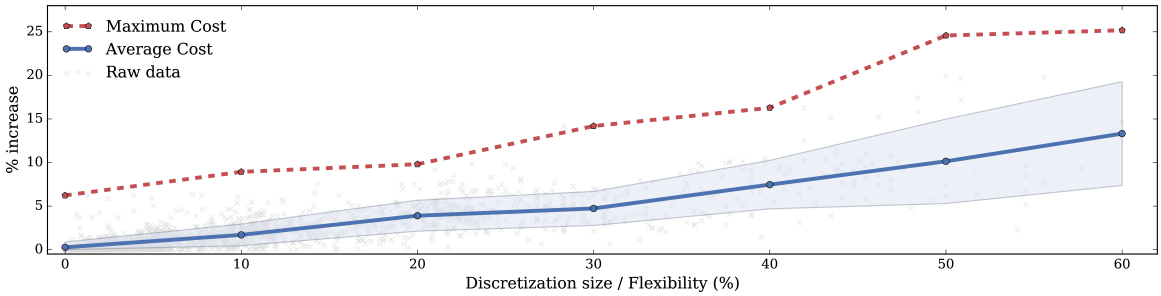


Figure 2.1: Cost increase due to discretization

The primary goal of this paper is to assess and quantify the price of discretizing time for a service network design problem, by means of an extensive computational study. The results will, of course, be specific to the service network design problem (and, too, to the specific instances used in the study), but we hope and expect that the findings are relevant also in other contexts where time-expanded network models are (or can be) used. The results show that the price of discretization can be high, with a loss of solution quality of more than 20% in some instances, see Figure 2.1 (note that the shaded area indicates the standard deviation of the cost increase). In the figure, instances are grouped based on the discretization scheme, i.e., the length of the time intervals, and a measure of flexibility. Flexibility is defined precisely, in the context of the service network design problem, in Section 2.4, but for now it can be considered as the extra available time, beyond the required, that is allowed in an instance. Intuitively, "flexible" instances should be less effected by coarse discretization schemes compared to "inflexible" instances, and so grouping by flexibility yields an informative comparison.

Even a small loss of solution quality can greatly impact the profitability of companies employing service network design models on a regular basis, such as parcel delivery

companies like UPS and FedEx, especially considering the size of their LTL revenue ($2.48 billion and $5.75 billion in 2015, respectfully), and railroad companies like Union Pacific ($21.81 billion in 2015) and BNSF ($21.97 billion in 2015). Moreover, in recent years there has been an increased emphasis on customer service, thus it is increasingly important to be aware of the loss of the missed opportunities that are introduced by coarse grained approximations.

Service network design is concerned with the trade-off between customer service and service cost; more specifically with the trade-off between route circuity and capacity utilization. Sending all freight along a shortest path from its origin to its destination results in low capacity utilization, but high customer satisfaction. Deviating from a shortest path (introducing route circuity) allows for freight consolidation and (if done well) an increase in capacity utilization. Since the transportation costs are (typically) based on truck or trailer miles, an optimal balance has to be found between route circuity (which increases truck miles) and capacity utilization (which decreases truck miles).

As noted by Magnanti and Wong, (1984), the mathematical model underlying service network design problems is similar to many other planning problems, reinforcing our belief that our results can provide meaningful insights for other problems that can be modeled using time-space networks, e.g., aircraft fleet assignment (Hane et al., [1995]), maritime inventory routing (Song and Furman, [2013]), railway crew scheduling (Vaidyanathan, Jha, and Ahuja, [2007]), multi-depot bus scheduling (Kliewer, Mellouli, and Suhl, [2006]), and military convoy planning (Chardaire et al., [2005]).

Even with a carefully chosen discretization, some time-expanded network models can get extremely large, and unable to be solved within acceptable computational limits, therefore researchers have developed heuristics to handle these instances. Teypaz, Schrenk, and Cung, (2010) propose a decomposition approach to tackle real-life sized service network design instances. Erera et al., (2012) develop an integer programming based large neighborhood search algorithm, utilizing a non-uniform time discretization, in which

exploring a neighborhood involves optimizing all paths for freight destined for a single terminal while keeping all other freight paths fixed. Bai et al., (2014) investigate a stochastic version of the service network design problem, advocating the use of a time-space network model, but acknowledging the difficulties encountered when solving the resulting large-scale mixed integer programs.

A related problem for which time-expanded network models and time-discretization strategies have been explored is the traveling salesman problem with time windows (TSPTW). Wang and Regan, (2002); Wang and Regan, (2009) introduce a discretization scheme that produces integer programs providing lower bounds and that has guaranteed convergence to the continuous-time optimal solution as the time interval size approaches zero. Dash et al., (2012) employ a similar idea, again in the context of TSPTW, and iteratively solve linear programming problems to determine a partition of the time windows (what they call *buckets*), as a preprocessing step in a branch-and-cut framework; they make the observation that the discretization strategy has a significant effect on the quality of the continuous-time approximation, and its balance with computational complexity.

Table 2.2: Selection of related papers using time-expanded networks

Paper	Interval size	Horizon
Inghels, Dullaert, and Vigo, (2016)	1 day	5 days
Neumann-Saavedra et al., (2016)	15 mins	1 day
Bai et al., (2014)	1 day	5 days
Song and Furman, (2013)	1 day	2 months
Erera et al., (2012)	2-6 hrs	1 week
Teypaz, Schrenk, and Cung, (2010)	1 period (21 mins)	480 periods (1 week)
Kobayashi and Kubo, (2010)	30 mins	5 weeks
Andersen, Crainic, and Christiansen, (2009)	2 hrs	1 week
Jarrah, Johnson, and Neubert, (2009)	1 day	1 week
Yan and Shih, (2007)	1 hr	196 hours
Nielsen et al., (2004)	12 hrs	30 days

Although the use of time-expanded networks is popular in both industry and in academia (see Table 2.2 for a small selection), to date, there has been little quantitative work that explores the price of discretizing time. Our computational study, in the context of service network design, changes that situation. It is based on 31 *flat* instances (i.e., that

specify a network and freight origins and destinations, but no time related information) which we use to generate 558 timed instances (18 for each flat instance) and 2790 time-discretized instances (5 discretizations for each timed instance – with 1, 5, 15, 30, and 60 minute time intervals; note that the average time horizon over all instances is 4489 minutes). The associated integer programs are solved using the Gurobi optimization software. The results, which took several months to compile, provide valuable insights into the price of discretizing time.

Having a better, quantitative understanding of the (potentially negative) impact of the choice of time discretization is informative and useful, and may prompt researchers and practitioners to be more careful when choosing a time discretization, but it does not, in and of itself, provide any mechanism to assist with that choice. Therefore, we also briefly examine and give problem characteristics and metrics that have predictive value and may help choose an appropriate time discretization (or at least prevent choosing a poor time discretization). As our results confirm, the primary loss in solution quality is due to the loss of cheap time-feasible paths, and the loss of potential consolidations. The results also show that instances with high flexibility are less effected by coarse discretization, but can be computationally difficult to solve especially when coupled with a high ratio of fixed and variable cost – since high flexibility allows for greater consolidation possibilities, while a high ratio of fixed and variable cost places emphasis on consolidations rather than routes.

The remainder of the paper is organized as follows. Section 2.2 gives a formal description of the service network design problem and its corresponding mixed integer programming model. Section 2.3 discusses the consequences of time discretization. Section 2.4 presents the design and results of our computational study. Section 2.5 examines the predictive value of metrics derived from the problem characteristics. Finally, we offer some final remarks and thoughts on future research in Section 2.6.

2.2 Problem Description

Service network design is a fundamental part of the freight transportation industry, helping make tactical decisions for profitability (Crainic, [2000]). This problem is NP-hard and solving reasonably sized instances to optimality is generally not practical. An early overview covering various algorithmic approaches is given by Magnanti and Wong, (1984), and more recently by Wieberneit, (2008). Most service network design research can be categorized (Hosseininasab, [2015]) as either focusing on real-life constraints, e.g., coordinating multiple fleet types (Andersen, Crainic, and Christiansen, [2009]), or, on algorithmic approaches to solve large instances, e.g., using branch-and-price (Andersen et al., [2011]).

In the service network design problem (SNDP) we are given a simple *flat* directed graph, $G = (N, A)$, and a set of commodities, K. The network G, has nodes, N, corresponding to terminals; and arcs, A, indicating that direct travel is possible between two terminals. Each arc $a = (n_1, n_2) \in A$ has an associated transit time τ_a, which is the duration of time to travel from node n_1 to n_2; a variable price v_a, which is the cost per quantity of a commodity sent along the arc (for example \$ per ton); a fixed price f_a, which is charged for every dispatch along the arc; and a dispatch capacity u_a, indicating the maximum consolidation quantity of each dispatch along this arc. Every commodity k in the set of commodities K must be routed from its origin $o^k \in N$ to its destination $d^k \in N$ along a unique path; has size q^k; and time-window $\left[e^k, l^k\right]$, representing the time the commodity becomes available for dispatch and the time by which it needs to reach its destination. The goal is to find a time-feasible origin-destination path for each commodity, and associated dispatch times at each of the nodes along the path, consolidating freight such that the total cost (i.e., incurred fixed and variable costs) is minimized. We assume that all parameters involving time, i.e., τ_a for $a \in A$, and e^k and l^k for $k \in K$, are nonnegative integers, sharing the same unit of time, and are known with certainty.

Without loss of generality, let $\min_{k \in K}\{e^k\} = 0$ and let the planning horizon, H, sharing the same time-units above, be such that $H \geq \max_{k \in K}\{l^k\}$. As mentioned previously, *uniform* time discretization partitions the planning horizon into intervals of equal length, say $\Delta \in \mathbb{R}$. The set of time points is then given by $\mathcal{T} := \{0, 1, \ldots, \lceil H/\Delta \rceil = T\}$. The associated time-expanded network, $\mathcal{G}_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{A}_\mathcal{T})$, has timed nodes $(n, t) \in \mathcal{N}_\mathcal{T}$ for $n \in N$ and $t \in \mathcal{T}$, and timed arcs $((n_1, t_1), (n_2, t_2)) \in \mathcal{A}_\mathcal{T}$ for $(n_1, n_2) \in A$ with $t_2 = t_1 + \lceil \tau_{(n_1,n_2)}/\Delta \rceil$ (i.e., conservative rounding scheme), for $0 \leq t_1 \leq t_2 \leq T$. In addition, there are *holding arcs*, which allow a commodity to stay at the same node for one time period, $((n, t), (n, t + 1)) \in \mathcal{A}_\mathcal{T}$ for $n \in N$ and $t = 0, \ldots, T - 1$. The time window of commodity $k \in K$ is mapped to $[\lceil e^k/\Delta \rceil, \lfloor l^k/\Delta \rfloor]$, that is, the time window shrinks according to the conservative rounding scheme.

Figures 2.2 and 2.3 give an example of conservative rounding on a small triangular network ($\tau_{1,0} = 9$, $\tau_{1,2} = 11$, $\tau_{0,2} = 4$) with a single commodity going from node 1 to node 2, with time window $[2, 19]$. The time horizon has been partitioned into time intervals of size Δ. When $\Delta = 3$, in Figure 2.3, the commodity's time window shrinks to $[3, 18]$ and the network transit times become $\tau_{1,0} = 9$, $\tau_{1,2} = 12$, $\tau_{0,2} = 6$.

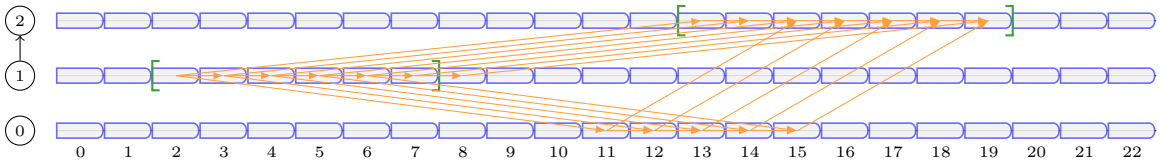


Figure 2.2: Example with $\Delta = 1$

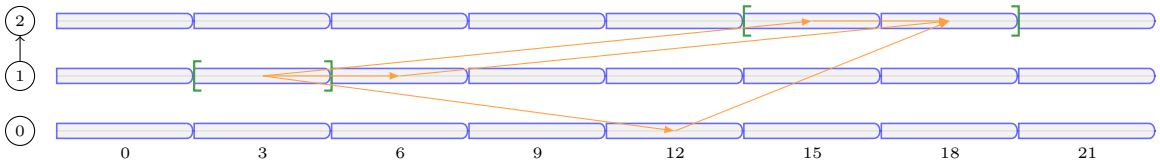


Figure 2.3: Example with $\Delta = 3$

Putting all this together gives the following time-space network model for SNDP:

Sets

K	commodities
$\mathcal{T}$	time points
$\mathcal{N}_\mathcal{T}$	nodes in the time-space network
$\mathcal{A}_\mathcal{T}$	arcs in the time-space network
$\delta_k^-(n,t)$	arcs into $(n,t) \in \mathcal{N}_\mathcal{T}$ for commodity $k \in K$
$\delta_k^+(n,t)$	arcs out of $(n,t) \in \mathcal{N}_\mathcal{T}$ for commodity $k \in K$

Parameters

o^k	origin node ($o^k \in N$) for commodity $k \in K$
d^k	destination node ($d^k \in N$) for commodity $k \in K$
e^k	earliest departure for commodity $k \in K$
l^k	latest arrival for commodity $k \in K$
τ_a	transit time along arc $a \in \mathcal{A}_\mathcal{T}$ (in time units)
f_a	fixed cost of arc $a \in \mathcal{A}_\mathcal{T}$ (\$ per dispatch)
v_a	variable cost of arc $a \in \mathcal{A}_\mathcal{T}$ (\$ per quantity)
u_a	capacity of arc $a \in \mathcal{A}_\mathcal{T}$ (in quantity units)

Decision Variables

$$x_a^k \quad \begin{cases} 1 & \text{if commodity } k \in K \text{ dispatches on arc } a \in \mathcal{A}_\mathcal{T} \\ 0 & \text{otherwise} \end{cases}$$

z_a integer quantity of all dispatches on arc $a \in \mathcal{A}_\mathcal{T}$

Model

$$C(\Delta) = \min \sum_{k \in K} \sum_{a \in \mathcal{A}_\mathcal{T}} v_a q^k x_a^k + \sum_{a \in \mathcal{A}_\mathcal{T}} f_a z_a \tag{2.1a}$$

s.t.

$$\sum_{a \in \delta_k^+(n,t)} x_a^k - \sum_{a \in \delta_k^-(n,t)} x_a^k = \begin{cases} 1 & n = o^k, t = \lceil e^k / \Delta \rceil \\ -1 & n = d^k, t = \lfloor l^k / \Delta \rfloor \\ 0 & \text{otherwise} \end{cases}, \quad \forall k \in K, (n,t) \in \mathcal{N}_\mathcal{T} \tag{2.1b}$$

$$\sum_{k \in K} q^k x_a^k \leq u_a z_a, \qquad \forall a \in \mathcal{A}_\mathcal{T} \tag{2.1c}$$

$$x_a^k \in \{0,1\}, \qquad \forall k \in K, a \in \mathcal{A}_\mathcal{T} \tag{2.1d}$$

$$z_a \in \mathbb{Z}_+, \qquad \forall a \in \mathcal{A}_\mathcal{T} \tag{2.1e}$$

The objective (2.1a) minimizes the total cost of transport, incorporating the variable and fixed dispatch costs. The flow constraints (2.1b) ensure that each commodity leaves its origin node after e^k and reaches its destination node before l^k, and constructs a path between them. Consolidation constraints (2.1c) force any commodities that travel on the same timed-arc to be dispatched together in multiples of capacity u_a. Lastly (2.1d) and (2.1e) define the variables and their associated domains.

2.3 Consequences of Discretization

Changes in solution quality are driven by two factors: the loss of low-cost feasible paths and the loss of consolidations. To fully appreciate the effect of discretization, it is useful to see what happens with a single instance for different values of the discretization parameter Δ. Figure 2.4 shows the *discretization gap* (DGap) as a function of the discretization parameter Δ, where the discretization gap is defined to be the continuous-time relative error $\frac{C(\Delta)-C(1)}{C(1)}$.

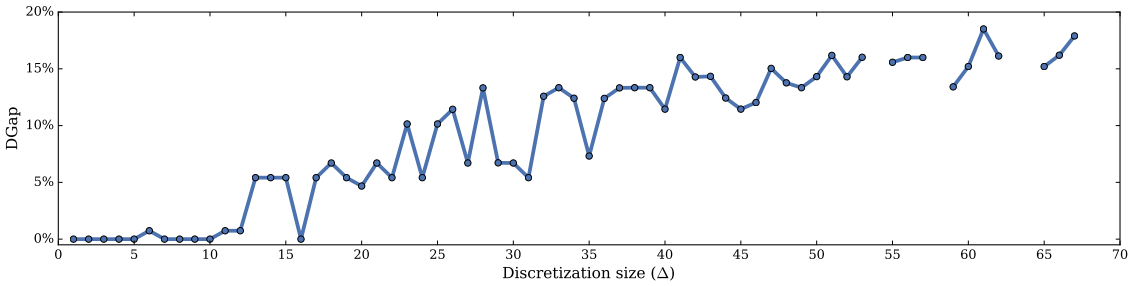
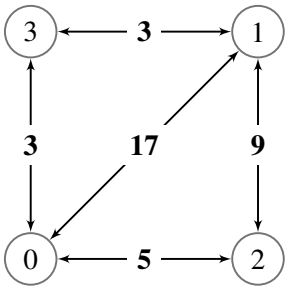


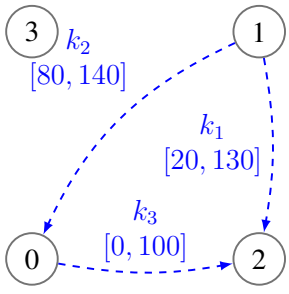
Figure 2.4: DGap vs Δ for a single instance

As expected, the discretization gap tends to increase with Δ, however, maybe somewhat surprisingly, the function is not monotonically non-decreasing and has missing values (towards the right side of the graph) due to infeasibility, i.e., at least one commodity has no time-feasible path from origin to destination. Note that it is possible to go from feasible to infeasible, and then to feasible again. These non-monotonic changes to solution quality, and infeasible points are illustrated by the following example.

Consider the network with the travel times shown in Figure 2.5a, where the fixed costs are equal to the travel times ($f_a = \tau_a$), the variable costs are zero ($v_a = 0$), and the capacity is three ($u_a = 3$). The commodities are shown in Figure 2.5b by means of dashed arcs, with the time window listed next to the arc. Each commodity has size 1 ($q^k = 1$), so that all commodities could consolidate if it is time-feasible.



(a) Network



(b) Commodities

Δ	3↔1	0↔2	0↔1	3↔0	1↔2
1	3	5	17	3	9
6	1	2	3	1	2
24	1	1	1	1	1
27	1	1	1	1	1
29	1	1	1	1	1
36	1	1	1	1	1
40	1	1	1	1	1

(c) Network τ_a Rounding

Figure 2.5: Example problem

Figure 2.6 shows the % cost increase due to Δ. Table 2.5c and Table 2.6 present transit time and solution details for a few specific time discretizations, respectively. From

this figure and the tables it can be easily seen how the cost changes due to the loss of consolidations and feasible paths. It also demonstrates how an instance can regain feasibility for larger discretizations, due to the nature of the conservative rounding scheme.

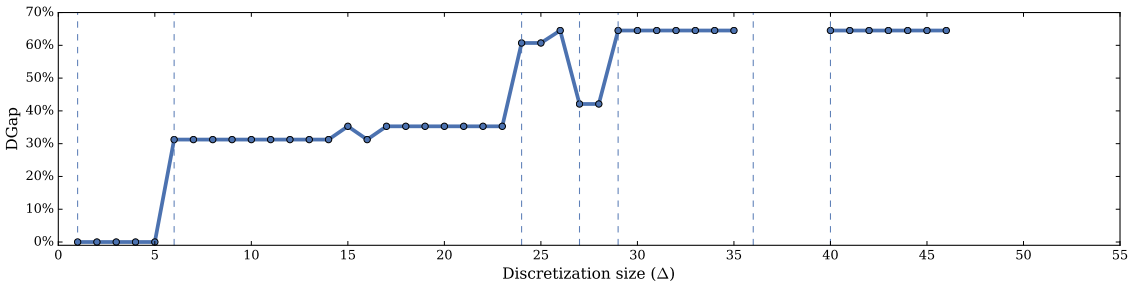


Figure 2.6: DGap vs Δ for Example Problem

Table 2.6: Events

Δ	$[e^{k_1}, l^{k_1}]$	$[e^{k_2}, l^{k_2}]$	$[e^{k_3}, l^{k_3}]$	k_1 Path	k_2 Path	k_3 Path	Consolidation	Cost	Description
1	[20,130]	[80,140]	[0,100]	$1,3,0,2$	$1,3,0$	$0,2$	$(1,3,0)\colon \{k_1,k_2\}, (0,2)\colon \{k_1,k_3\}$	11	Optimal solution
6	[4,21]	[14,23]	[0,16]	$1,3,0,2$	$1,3,0$	$0,2$	$(1,3,0)\colon \{k_1,k_2\}$	16	$\{k_1,k_3\}$ consolidation lost
24	[1,5]	[4,5]	[0,4]	$1,3,0,2$	$1,0$	$0,2$	$(0,2)\colon \{k_1,k_3\}$	28	$\{k_1,k_2\}$ consolidation lost
27	[1,4]	[3,5]	[0,3]	$1,2$	$1,2,0$	$0,2$	$(1,3)\colon \{k_1,k_2\}$	19	k_2 cheaper feasible path
29	[1,4]	[3,4]	[0,3]	$1,2$	$1,0$	$0,2$		31	$\{k_1,k_2\}$ consolidation lost
36	[1,3]	[3,3]	[0,2]					-	k_2 has no feasible path
40	[1,3]	[2,3]	[0,2]	$1,2$	$1,0$	$0,2$		31	k_2 regains feasible path

2.4 Computational Study

Since all parameters that involve time are given as integers, choosing $\Delta = 1$ results in a time-expanded network model that yields a continuous-time optimal solution (when solved to optimality). It is often challenging to choose a Δ that induces both a tractable IP model, and one that has a small discretization gap. A pragmatic alternative is to choose a fine discretization (i.e., Δ close to 1), and enforce a computation time limit (e.g., of 3 hours). Let $\widetilde{C}(\Delta)$ be the cost of the best solution found within this time limit, then we refer to $\frac{\widetilde{C}(\Delta)-C(1)}{C(1)}$ as the *early termination discretization gap* (ETDGap). Our analysis of the price of discretization is a quantitative investigation of both the DGap and the ETDGap.

2.4.1 Instance Generation

Our computational study uses the timed instances of SNDP generated by Boland et al., (2017), in which a time component is added to the C instances introduced by Crainic, Frangioni, and Gendron, (2001). The C instances have been used as a benchmark for a number of algorithms for the capacitated fixed charge network design problem, and have been randomly generated to cover a range of parameters. Each class in Table 2.7 corresponds to a flat instance with network $G = (N, A)$ and set of commodities K. Each commodity $k \in K$ has an origin $o^k \in N$, destination $d^k \in N$, and quantity $q^k \in \mathbb{R}$. Note that the fixed and variable costs, as well as commodity quantities and arc capacities have been randomly generated such that their ratios $\frac{1}{|A|}\sum_{a\in A}\frac{f_a}{v_a u_a}$ (cost ratio), and $\sum_{k\in K} q^k / \frac{1}{|A|}\sum_{a\in A} u_a$ (capacity ratio) approximate a certain target (see Crainic, Frangioni, and Gendron, (2001) for more details); the actual ratios are given in Table 2.7.

Timed instances are generated by first calculating the transit times τ_a for each arc $a \in A$ by assuming the fixed cost of travel f_a represents 55¢ per mile, and that travel takes place at 60 miles per hour, i.e., $\tau_a = f_a / 0.55$ (minutes). Let $\gamma_1(o^k, d^k)$ be the length of the shortest path (i.e., least total transit time) from o^k to d^k for commodity $k \in K$, then the average shortest path length, $\mathcal{L} = 1/|K| \sum_{k\in K} \gamma_1(o^k, d^k)$, is used as a parameter to generate time windows $[e^k, l^k]$ for each commodity by randomly sampling from a normal distribution (see Table 2.8), where $l^k = e^k + \gamma_1(o^k, d^k) + f_1^k$, and f_1^k represents the flexibility of the time window.

Observe that 6 sets of time parameters are defined. For each of these sets and for each of the 31 flat instances, we create 3 realizations of the random parameters, giving 558 continuous-time instances. For each of these timed instances, we create 5 discretized instances of the time-expanded network model using discretization Δ = 1, 5, 15, 30, and 60. By our assumption of integer parameters, the discretized and continuous-time instances are equivalent when $\Delta = 1$, hence we will simply refer to both as *instances* unless clarification is required.

Table 2.7: Flat-instances from Crainic, Frangioni, and Gendron, (2001)

Class	$\|N\|$	$\|A\|$	$\|K\|$	Cost ratio	Cap ratio	$\mathcal{L}$
c33	20	228	39	0.02	5.8	2,407.9
c35	20	230	40	0.02	16.0	767.9
c36	20	230	40	0.08	16.0	3,705.8
c37	20	228	200	0.51	16.0	1,871.4
c38	20	230	200	0.97	16.0	4,381.0
c39	20	229	200	0.47	20.0	1,691.3
c40	20	228	200	0.94	22.0	3,522.1
c41	20	288	40	0.02	8.0	1,622.0
c42	20	294	40	0.08	10.0	5,675.8
c43	20	294	40	0.02	16.0	776.5
c44	20	294	40	0.08	16.0	3,517.9
c45	20	294	200	0.48	25.0	1,124.2
c46	20	292	200	1.01	25.0	2,632.0
c47	20	291	200	0.46	28.0	996.6
c48	20	291	200	0.95	28.0	2,271.6
c49	30	518	100	0.10	20.0	341.1
c50	30	516	100	0.51	20.0	1,586.5
c51	30	519	100	0.09	29.9	206.6
c52	30	517	100	0.49	29.9	1,161.5
c53	30	520	400	0.18	40.0	612.1
c54	30	520	400	0.36	40.0	1,061.8
c55	30	516	400	0.18	49.9	479.4
c56	30	518	400	0.35	49.9	966.9
c57	30	680	100	0.09	20.0	307.6
c58	30	680	100	0.20	20.0	592.8
c59	30	687	100	0.10	29.9	187.1
c60	30	686	100	0.20	29.9	394.7
c61	30	685	400	0.19	40.0	503.8
c62	30	679	400	0.36	40.0	1,056.5
c63	30	678	400	0.18	49.9	381.4
c64	30	683	400	0.34	49.9	780.0

Table 2.8: Time-oriented characteristics

#	$e^k \sim N(\mathcal{L}, \sigma)$	$f_1^k \sim N(\mu, \mathcal{L}/6)$
1	$\sigma = \mathcal{L}/3$	$\mu = \mathcal{L}/2$
2	$\sigma = \mathcal{L}/6$	$\mu = \mathcal{L}/2$
3	$\sigma = \mathcal{L}/9$	$\mu = \mathcal{L}/2$
4	$\sigma = \mathcal{L}/3$	$\mu = \mathcal{L}/4$
5	$\sigma = \mathcal{L}/6$	$\mu = \mathcal{L}/4$
6	$\sigma = \mathcal{L}/9$	$\mu = \mathcal{L}/4$

2.4.2 Results

All instances were solved using Gurobi 5.6.3 on a 32 core Opteron 6274 2200 MHz server, each with a 3 hour time limit, 24Gb RAM (dedicated, not machine limited), and 1% optimality gap tolerance. Gurobi successfully generated the associated IP for each instance, and could place each in memory. On average, IP model construction took around 215 seconds; but for 2% of the instances, it took longer than 30 minutes, but never more than an hour. Note that the 3 hour computational limit excludes model construction time. All instances were additionally solved again with $\Delta = 1$ (without computational limits), to find the continuous-time optimal value for the DGap and ETDGap calculations.

When using a discretization parameter $\Delta > 1$, travel times as well as early and late times have to be adjusted. As a result, the shortest origin-destination path for a commodity (in terms of travel time) in the aggregated time-expanded network may no longer be feasible, i.e., even when departing as early as possible at the origin, the arrival at the destination will occur after the latest allowable arrival time. If this happens with a commodity in an instance, the instance is labeled as infeasible (for the particular value of Δ). On the other hand, an origin-destination path in the aggregated time-expanded network may be time-feasible, but, when evaluated using the true travel times and the true early and late times, the path may no longer be time-feasible. If this happens with a commodity in an instance, the instance is labeled as non-implementable (for the particular value of Δ).

Recall that we choose to use conservative rounding when discretizing time. With conservative rounding, feasible solutions to the aggregated time-expanded network model are always implementable (for any Δ). That is, each dispatch for every commodity can be successfully executed when mapped back to continuous time (i.e., the dispatch is continuous-time feasible). When regular or optimistic rounding is used instead, this may no longer be the case. In fact, as shown in Table 2.9, which is based on a subset of 792 instances, far fewer implementable solutions are obtained when using regular or optimistic rounding. (Note that the presented results reflect whether the time-expanded network model

has a feasible solution, and, if so, whether that feasible solution is implementable.)

Table 2.9: Percentage of implementable instances for different rounding schemes (792 instances)

Discretization	Conservative	Regular	Optimistic
5 minutes	100.0	83.3	61.6
15 minutes	97.5	67.7	34.8
30 minutes	90.9	53.5	20.2
60 minutes	77.3	39.4	10.6

From this point on we will only be discussing results and instances that use the conservative rounding scheme. We begin our analysis of the 2790 instances by a relative frequency histogram of their solution status (Figure 2.7). Note that an instance can be infeasible (depending on the discretization parameter Δ), if it is feasible, then it can run out of memory, and, if it does not run out of memory, it can run out of time (optimality is not proved within the 3-hour time limit).

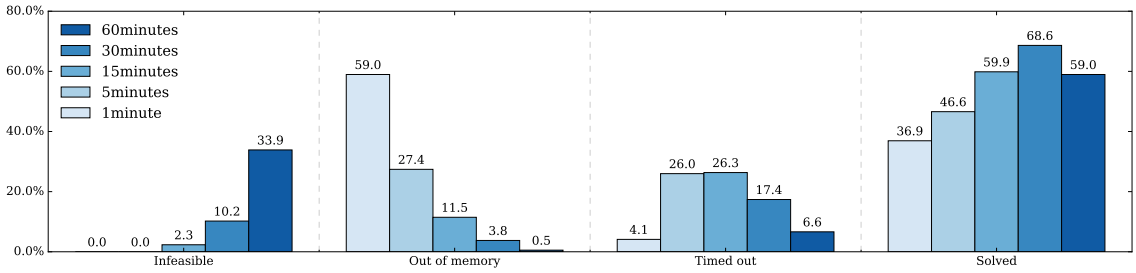


Figure 2.7: Solution status for instances for different discretization parameter values

We observe, as expected, that finer time discretizations, which induce larger IPs, are more likely to hit the memory or time limit, and that coarser time discretizations, which induce smaller IPs, are more likely to be infeasible. Note that the instances that timed out are confounded with those that run out of memory, that is, it is likely that many of these instances would time out if the memory limitation was not enforced.

Although all of our instances have the same time unit (minute) they are not necessarily directly comparable due to differences in the scale of the parameters and the network structure. In order for the results to provide meaningful insights, it is beneficial to group

instances together based on similar characteristics. Of particular interest is the *flexibility* of commodities, which relates the width of the time window of a commodity to the minimum travel time from a commodity's origin to its destination. When an instance has inflexible commodities, the route that a commodity follows from its origin to its destination cannot deviate much from the fastest path, and, thus, there are fewer opportunities for consolidation. Importantly, as Δ increases, the flexibility of a commodity is reduced due to the conservative rounding scheme. Consequently, inflexible instances are more effected by discretization than flexible instances.

Let $\gamma_\Delta(n_1, n_2)$ be the minimum travel time from node n_1 to n_2 with time discretization Δ, and let $\Gamma_\Delta(n_1, n_2)$ be its associated path, i.e., $\gamma_\Delta(n_1, n_2) = \sum_{a \in \Gamma_\Delta(n_1,n_2)} \lceil \tau_a / \Delta \rceil$. Then the flexibility of commodity k in a discretized instance is given by

$$f^k_\Delta = \lfloor l^k / \Delta \rfloor - \lceil e^k / \Delta \rceil - \gamma_\Delta(o^k, d^k).$$

We define the flexibility of a discretized instance as $f = \min_{k \in K} \{f^k_1\}$.

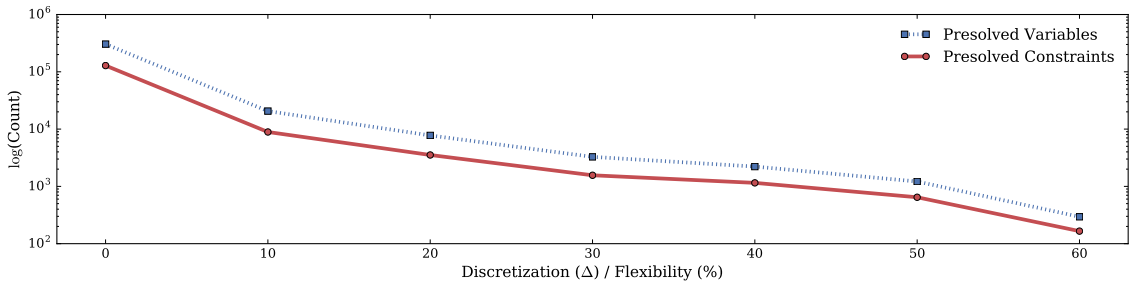


Figure 2.8: Average Model Size as a function of Δ/f

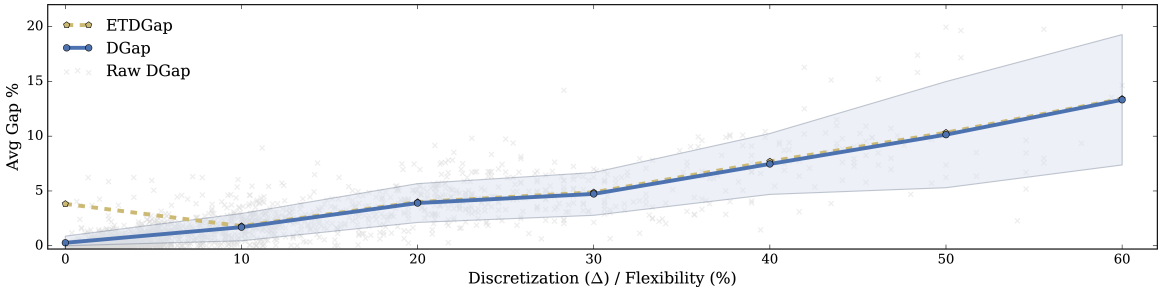


Figure 2.9: Average ETDGap and DGap per Δ/f

For the discretized instances with feasible solutions to the time-expanded network model, Figure 2.8 shows the average size of the associated IP in terms of the number of variables and constraints and Figure 2.9 shows the average DGap and the average ETDGap, as a function of the ratio of discretization and flexibility (rounded to the nearest ten percent in Figure 2.8). The figures clearly demonstrate the link between high quality approximations and computationally challenging IPs, that is, fine time discretizations yield computationally challenging IPs with high-quality approximations to the continuous-time problem, whereas coarse time discretizations yield computationally tractable IPs with low-quality approximations. Furthermore, for these instances, the imposed time limit of three hours does not seem to have any effect when the discretization is greater than 10% of the flexibility.

Next, we characterize flexibility in a slightly different way, based on the observation that the largest sensible choice for Δ for an instance is given by

$$\max\left\{\Delta \in \mathbb{Z}_+ : \frac{\lfloor l^k/\Delta \rfloor - \lceil e^k/\Delta \rceil}{\gamma_\Delta(o^k, d^k)} \geq 1, \quad \forall\, k \in K\right\},$$

because when the value of the ratio drops below one, then the instance becomes infeasible. By thinking of the fractional part of any value as being a uniform random variable $U[0,1]$, considering the equation

$$1 = \frac{\lfloor l^k/\Delta \rfloor - \lceil e^k/\Delta \rceil}{\gamma_\Delta(o^k, d^k)} \approx \frac{\mathbb{E}\left[(l^k/\Delta - U_l) - (e^k/\Delta + U_e)\right]}{\mathbb{E}\left[\sum_{a \in \Gamma_\Delta(o^k, d^k)} (\tau_a/\Delta + U_a)\right]} \approx \frac{\frac{l^k - e^k}{\Delta} - 1}{\frac{\gamma_1(o^k, d^k)}{\Delta} + \frac{1}{2}|\Gamma_1(o^k, d^k)|},$$

and solving for Δ over all $k \in K$ gives

$$\Delta_{max} = \left\lfloor \min_{k \in K} \left\{ \frac{l^k - e^k - \gamma_1(o^k, d^k)}{1 + \frac{1}{2}|\Gamma_1(o^k, d^k)|} \right\} \right\rfloor.$$

Figure 2.10 shows the results grouped by Δ/Δ_{max}. We observe that the DGap average and variance is quite stable up until $\Delta/\Delta_{max} = 1$, where it then starts to wildly fluctuate as many of the instances become infeasible. This highlights the loss of quality due to increasing inflexibility – which is the critical factor in the price of discretization. We note that it is unlikely for a practitioner to voluntarily choose a discretization with $\Delta/\Delta_{max} > 1$,

and thus for the most part these points can be ignored.

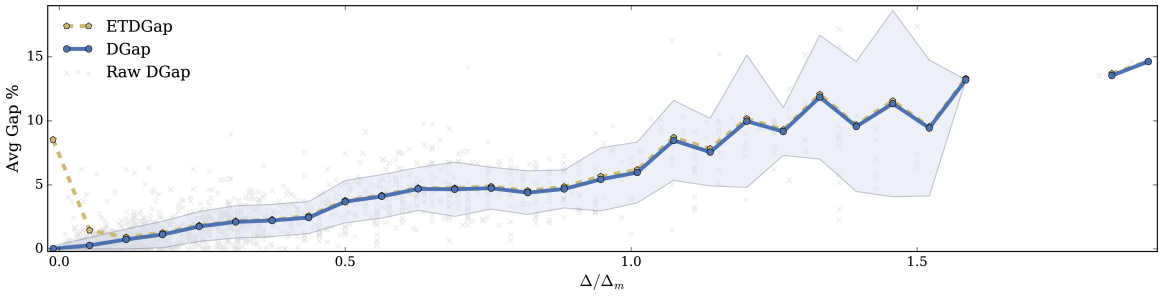


Figure 2.10: Average ETDGap / DGap as a function of Δ/Δ_{max}

Notice again that the ETDGap does not look particularly significant in Figure 2.10, however, in Figures 2.11 and 2.12 we observe that instances with a large number of commodities and a large cost ratio (i.e., high fixed costs and low variable costs), have a large average ETDGap. A high cost ratio puts extra emphasis on finding (low-cost) consolidations, and a large number of commodities increases the possibilities for consolidations; and so instances with both these characteristics are typically harder to solve and thus more likely to terminate early with a low-quality solution. In particular, notice that in Table 2.7 the instances with the highest Cost Ratio have $|K| = 200$, and observe its high ETDGap in Figure 2.11. Out of the 716 feasible instances with $|K| = 200$, 41.2% ran out of memory, and 33.1% timed-out.

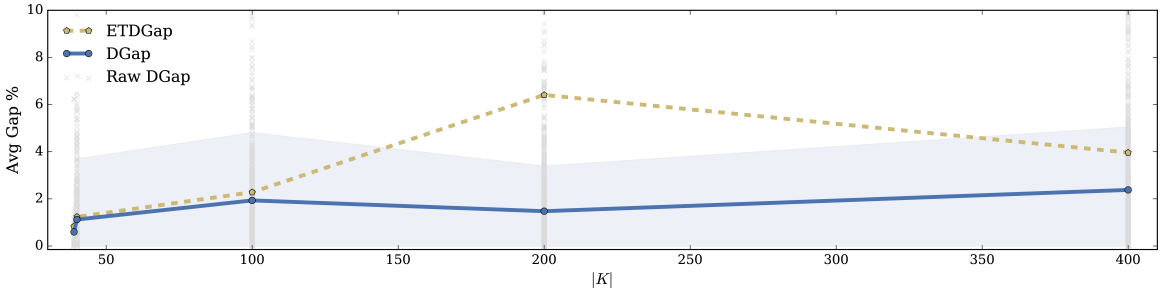


Figure 2.11: Average ETDGap / DGap as a function of $|K|$

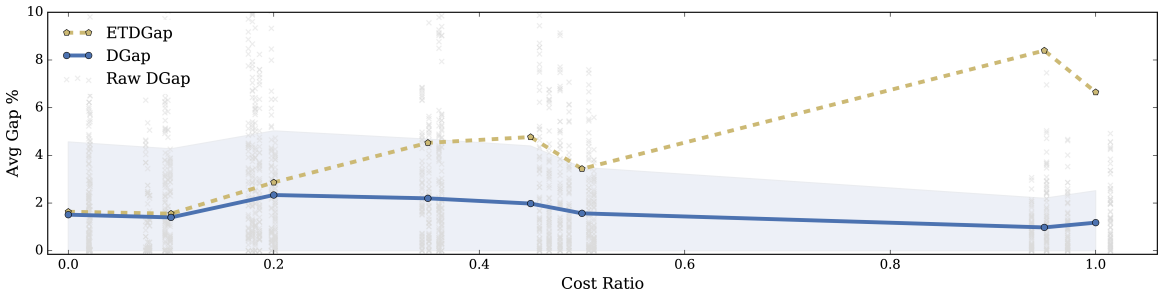


Figure 2.12: Average ETDGap / DGap as a function of the cost ratio

2.5 Choosing a Discretization Size / Estimating DGap

The graph in Figure 2.10 relates the discretization gap for an instance to the discretization parameter. This can be used to derive for a desired maximum discretization gap $\overline{DGap}$, a reasonable discretization choice:

$$\Delta = \min \left\{ \left\lfloor \frac{0.65}{5} \, \Delta_{max} \, \overline{DGap} \right\rfloor, \Delta_{max} \right\}, \tag{2.2}$$

where we have used the fact that for $0.65 \leq \Delta/\Delta_{max} \leq 1$, the graph is almost flat.

Alternatively, it is possible to quickly estimate the DGap for a particular instance and a given Δ, by exploring the observations of Section 2.3. In particular, as Δ increases, solution quality drops due to a loss of low-cost feasible paths. It is possible to estimate this loss using the cost of the cheapest time-feasible origin destination path for each commodity k, which we denote by ρ^k_Δ (using $v_a q^k + f_a \lceil \frac{q^k}{u_a} \rceil$ as the cost of traversing arc $a \in A$). Obtaining this cost involves solving an elementary resource constrained shortest path problem, which is NP-hard (Dror, [1994]), but in practice relatively easy (especially since we are using the flat network rather than the time-expanded network). Many special purpose algorithms have been designed for its solution, e.g., Feillet et al. (2004), Boland, Dethridge, and Dumitrescu (2006), and Garcia (2009). Given discretization parameter Δ, we can estimate the discretization gap to be

$$\frac{\sum_{k \in K} \rho^k_\Delta}{\sum_{k \in K} \rho^k_1}.$$

Since this only takes the loss of low-cost time-feasible paths into consideration, this is an

under-estimate. See Figure 2.13 for a comparison between this estimate and the actual. Note that the estimate is more accurate for instances with low cost ratios (Figure 2.14), in which path cost dominates consolidation benefits.

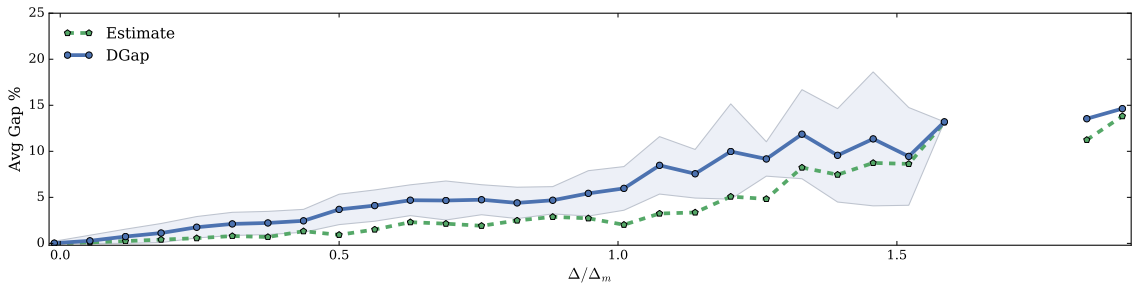


Figure 2.13: Estimate and Actual as a function of Δ/Δ_{max}

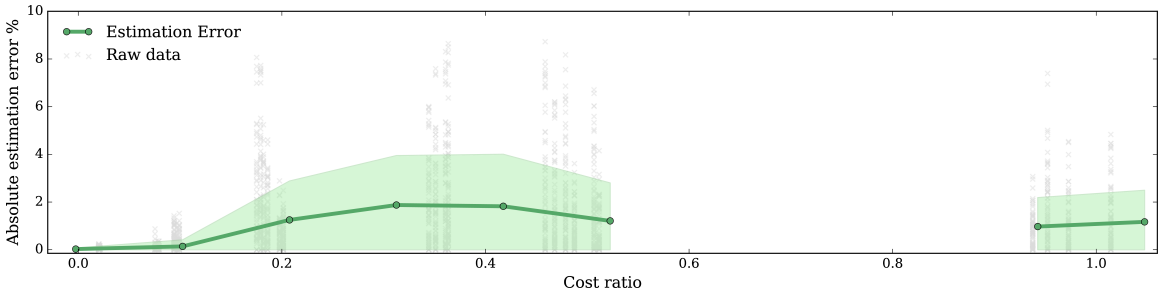


Figure 2.14: Estimation Error vs cost ratio

2.6 Conclusion

Time-expanded networks are popular and useful when modeling transportation problems, however their use requires that a discretization scheme is chosen. Our empirical results show that this is an important choice (at least in the context of the service network design problem), since the price of discretization can be high.

Most commonly, *uniform* discretization is used, which exhibits a trade-off between solution quality and tractability; making the choice for a suitable interval size challenging. To aid in this decision, we have provided metrics that estimate the price of discretization, and also provided a mechanism to choose an interval size for a given tolerance. In particular we have shown that it is possible, again in the context of the service network design

problem, to develop metrics that can help identify, a priori, instances that are sensitive to a coarse discretization. We believe that the insights resulting from our computational study are also relevant in other transportation settings (that are naturally modeled using time-expanded networks), and therefore hope that we have provided value to practitioners and researchers alike.

We have shown that the primary loss in solution quality is due to the loss of low-cost time-feasible paths, and the loss of potential consolidations. Instances with high flexibility are less effected by coarse discretization, but can be difficult to solve, especially with high cost ratios - since high flexibility allows for greater consolidation possibilities and a high cost ratio places emphasis on consolidations over routes. In particular to SNDP, instances with low cost ratios can be approximated by the elementary resource constrained shortest path problem, since route cost dominates the benefits of consolidation.

Our study has been restricted to *uniform* time discretization schemes, which partition the planning horizon into equal-length time intervals. Schemes that are *non-uniform* and derived from the structural properties of optimal schedules can provide a powerful alternative, and could potentially avoid the trade-off between solution quality and tractability. More research into these discretization schemes is certainly needed. Currently, the most promising option may be to use dynamic discretization discovery methods, which start from an extremely coarse non-uniform time discretization and iteratively refine the time discretization until a continuous-time optimal schedule is found without ever generating the complete time-expanded network model; see Boland et al., (2017) for such an approach.

CHAPTER 3
DYNAMIC DISCRETIZATION DISCOVERY

3.1 Introduction

Consolidation carriers transport shipments that are small relative to trailer capacity. Such shipments are vital to e-Commerce. Consolidation carriers operate in (1) the less-than-truckload (LTL) freight transport sector, a sector with annual revenues in the United States alone of about $30 billion (Schulz, [2014]), and (2) the small package/parcel transport sector, a sector with much larger annual revenues, with one player alone (UPS) reporting $54 billion in revenue in 2012. Both LTL and small package carriers play a prominent role in the fulfillment of orders placed online (as well as other channels). Fast shipping times (and low cost) are critical to the success of the online sales channel, and e-tailers, such as Amazon.com, are continuously pushing the boundary, aiming for next-day and even same-day delivery. These trends result in increased pressure on LTL and small package transport companies to deliver in less time (without increasing their cost).

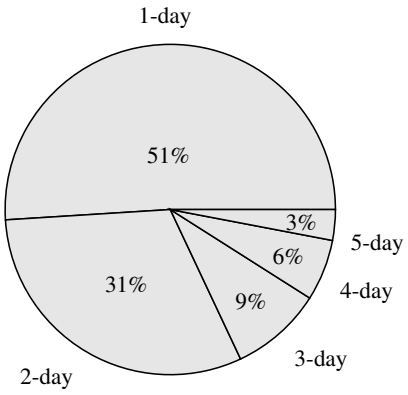


Figure 3.1: Freight profile for a large LTL carrier by service

This phenomenon is reflected in Figure 3.1, which shows the freight profile for a large LTL carrier by service level. It shows that over 80% of their shipments need to be delivered

within two days.

To deliver goods in a cost-effective manner, a consolidation carrier must consolidate shipments, which requires coordinating the paths for different shipments in both space and time. The push towards rapid delivery reduces the margin for error in this coordination, which necessitates planning processes that accurately time dispatches. These planning processes have long been supported by solving the so-called *Service Network Design* problem (Crainic, [2000]; Wieberneit, [2008]), which decides the paths for the shipments and the services (or resources) necessary to execute them. Service network design decisions for a consolidation carrier have both a geographic and temporal component, e.g., "dispatch a truck from Chicago, IL to Atlanta, GA at 9.05 pm." A common technique for modeling the temporal component is discretization; instead of deciding the exact time at which a dispatch should occur (e.g., 7.38 pm), the model decides a time interval during which the dispatch should occur (e.g., between 6pm and 8 pm).

When discretizing time, service network design problems can be formulated on a time-expanded network (Ford and Fulkerson, [1958]; Ford and Fulkerson, [1962]), in which a node encodes both a location and a time interval, and solutions prescribe dispatch time intervals for resources (trucks, drivers, etc.) and shipments. Service network design models calculate the costs for a set of dispatch decisions by estimating consolidation opportunities, i.e., by recognizing that prescribed dispatch time intervals for shipments allow travel together using the same resource. For example, shipments that should dispatch from the same origin node to the same destination node in the same dispatch time interval (say from Louisville, KY to Jackson, MI between 6 and 11 pm) are candidates for consolidation.

Clearly, the granularity of the time discretization has an impact on the candidate consolidation opportunities identified. At the same time, the granularity of the time discretization also impacts the computational tractability. With an hourly discretization of a week-long planning horizon, 168 timed copies of a node representing a location will be created. With a 5-minute discretization of a week-long planning horizon, 2,016 timed

copies of a node representing a location will be created. The latter discretization will likely yield a service network design problem that is much too large to fit into memory or solve in a reasonable amount of time. (In his introduction to network flows over time Skutella (2009) also notes that the use of a discretization that includes each possibly relevant time point can be challenging computationally in many problem settings.)

While there is widespread use of discretizations of time and time-expanded networks in service network design models (Jarrah, Johnson, and Neubert, [2009]; Andersen et al., [2011]; Erera et al., [2012]; Crainic et al., [2014]), we postulate that the fundamental question related to their use has not yet been answered: **Is it possible to produce an optimal "continuous" time solution without explicitly modeling each point in time**? In this paper, we show that this question can be answered in the affirmative. We refer to a service network design problem in which time is modeled in such a way that it accurately captures the consolidation opportunities as a Continuous Time Service Network Design Problem (CTSNDP). For all practical purposes, a time-expanded network based on a 1-minute time discretization gives a CTSNDP. (Therefore, in the remainder, we will assume that the travel times and the times at which commodities become available and are due are specified as integers.) Furthermore, we call a time-expanded network that does not include all the time points a *partially time-expanded network*.

We develop a *dynamic discretization discovery algorithm* that manipulates partially time-expanded networks and allows the solution of a CTSNDP without ever creating a fully time-expanded network. The algorithm repeatedly solves a service network design problem defined on a partially time-expanded network and refines the partially time-expanded network based on an analysis of the solution obtained. Each partially time-expanded network is such that the resulting service network design problem is a relaxation of the CTSNDP. Furthermore, the solution to this relaxation can be converted to a feasible solution to the CTSNDP by solving an appropriately defined linear program. If the converted (or repaired) solution has the same cost, it will be optimal. If not, then the

linear programming solution identifies time points that can be added to the partially time-expanded network to ensure that an improved solution is obtained in the next iteration. A flow chart of the high-level structure of the dynamic discretization discovery algorithm can be found in Figure 3.2. Thus, the dynamic discretization discovery algorithm solves a sequence of small MIPs, rather than a single large MIP.

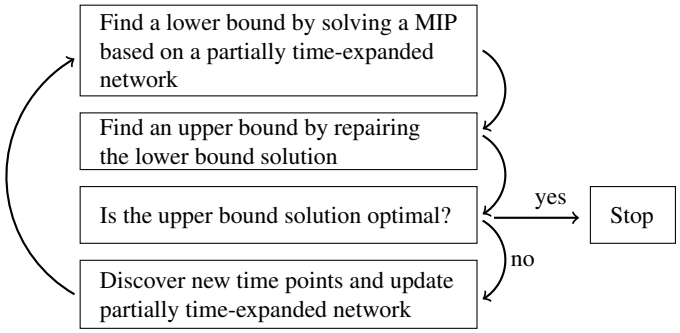


Figure 3.2: Flow chart of a dynamic discretization discovery algorithm.

An extensive computational study shows the efficacy of the algorithm: instances with networks consisting of 30 nodes and 700 arcs, with 400 commodities, and a planning horizon of about 8 hours, which, when using a full time discretization of 1 minute, leads to integer programs with more than 1,500,000 variables and close to 1,400,000 constraints, can often be solved to proven optimality in less than 30 minutes. Furthermore, the algorithm solves 97% of the several hundred instances in our test set and does so, on average, in less than 15 minutes. For those it does not solve the algorithm produces, on average, a solution with a provable optimality gap of 2.5% or less in two hours. Computational results on a few instances derived from data from a real-world less-than-truckload carrier are also promising with high-quality solutions produced in two hours or less.

To summarize, the main contributions of the paper are (1) the development of an algorithm for efficiently solving a continuous time service network design problem, and (2) demonstrating that an optimization problem defined on a time-expanded network can be solved to proven optimality without ever generating the complete time-expanded network.

As time-expanded networks are frequently used to model transportation problems, we hope that the latter will stimulate other researchers to explore similar approaches in other contexts, and that that will ultimately result in an improved ability to solve practically relevant problems.

The remainder of the paper is organized as follows. In Section 3.2, we review relevant literature. In Section 3.3, we present a formal description of the CTSNDP and discuss a property that (to some extent) motivates our approach. In Section 3.4, we introduce an iterative refinement algorithm for solving CTSNDP. In Section 3.5, we present and interpret the results of an extensive computational study of the algorithm's performance. Finally, in Section 3.6, we finish with conclusions and a discussion of future work.

3.2 Literature review

The importance of incorporating temporal aspects into flow models has been recognized since their inception. Already in 1958, Ford and Fulkerson (1958) introduced the notion of *flows over time* (also called *dynamic flows*). They considered networks with transit times on the arcs, specifying the amount of time it takes for flow to travel from the tail of the arc to the head of the arc, and sought to send a maximum flow from a source to a sink within a given time horizon. They showed that a flows-over-time problem in a network with transit times can be converted to an equivalent standard (static) flow problem in a corresponding time-expanded network. The fundamental concept of an s-t-cut in a network was extended to an s-t-cut over time as well (Anderson, Nash, and Philpott, [1982]; Anderson and Nash, [1987]). A comprehensive overview of this research area can be found in Skutella, (2009).

Similarly, researchers have extended the minimum cost s-t-flow problem to include a temporal component. Klinz and Woeginger (2004) show that, unlike the static problem, the minimum cost s-t-flow over time problem is weakly NP-Hard. Fleischer and Skutella (2007) provide a polynomial time approximation scheme for this (and other) problems (see also Fleischer and Skutella (2003)).

A problem that is more closely related to the CTSNDP is the multi-commodity flow over time problem (Hall, Hippler, and Skutella, [2007]), in which demands must be routed from sources to sinks within a given time horizon. Hall, Hippler, and Skutella (2007) characterizes when this problem is weakly NP-Hard. Topaloglu and Powell (2006) study a time-staged stochastic integer multi-commodity flow problem.

The problems mentioned above assume a fixed time horizon is provided as part of the input. Researchers have also looked at flow models where the objective is to minimize the time it takes to send a given amount of flow. For example, Burkard, Dlaska, and Klinz, (1993) present an algorithm that solves the quickest s-t flow problem in strongly polynomial time. Similarly, Hoppe and Tardos, (2000) provide a polynomial-time algorithm to solve the quickest transshipment problem. Researchers have also studied the quickest multi-commodity flow problem, for which Fleischer and Skutella, (2007) provide an approximation algorithm with performance guarantee of 2. Researchers have also studied problems that seek flows with an earliest arrival property, in which the flows arriving at the destination at each time point are maximized (Gale, [1958]; Minieka, [1973]; Megiddo, [1974]; Jarvis and Ratliff, [1982]; Hoppe and Tardos, [1994]; Tjandra, [2003]; Baumann and Skutella, [2006]).

The CTSNDP adds an additional layer of complexity to the multi-commodity flow over time problem by also incorporating network design decisions, which introduces a packing component to the problem. Kennington and Nicholson (2010) study a related problem – the uncapacitated fixed-charge network flow problem defined on a time-expanded network – but focus on choosing appropriate artificial capacities on the arcs to strengthen the linear programming relaxation of the natural integer programming formulation, and only consider instances with relatively small time-expanded networks. Fischer and Helmberg, (2012) develop methods for dynamically generating time-expanded networks, but do so in the context of solving shortest path problems, and without having to make design decisions.

Powell, Jaillet, and Odoni, (1995) discuss the use of time-expanded networks in

logistics planning models, noting that (at that time) most models create a time-expanded network by simply replicating the underlying network each time period.

Research on using partial time discretizations and dynamically adjusting a time discretization is scarce. Fleischer and Skutella, (2007) use partial discretizations to generate (near-)optimal solutions to quickest-flow-over-time problems. (They refer to a partially time-expanded network as a *condensed* time-expanded network.) Wang and Regan (2009) analyze the convergence of a time window discretization method for the traveling salesman problem with time windows (TSPTW) introduced by Wang and Regan (2002) to obtain lower bounds on the optimal value. Their analysis shows that iteratively refining the discretization converges to the optimal value. Dash et al. (2012) present an extended formulation for the TSPTW based on partitioning the time windows into subwindows called buckets (which can be thought of as discretizing the time window). They present cutting planes for this formulation that are computationally more effective than the ones known in the literature because they exploit the division of the time windows into buckets. They propose an iterative refinement scheme to determine appropriate partitions of the time windows. We provide more detail on the similarities and differences between our method and that of Dash et al. (2012) in Section 3.4.5.

Unlike the quickest-flow-over-time problems mentioned above, optimal solutions to CTSNDP need to strike a balance between the flow time from origin to destination and the capacity utilization of the arcs in the network, with flows waiting at the tail of an arc to be consolidated with other flows using the same arc. Furthermore, the continuous time flow models described above do not explicitly capture the delivery time constraints encountered in many transportation problems. To the best of our knowledge, we are the first to look at dynamically generating a (partially) time-expanded network for a problem that captures design decisions as well as flow time windows.

3.3 Problem description

Let $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ be a network with node set $\mathcal{N}$ and directed arc set $\mathcal{A}$. We will often refer to $\mathcal{D}$ as a "flat" network, as opposed to a time-expanded network, because the nodes in $\mathcal{N}$ model physical locations. Associated with each arc $a = (i, j) \in \mathcal{A}$ is a travel time $\tau_{ij} \in \mathbb{N}_{>0}$, a per-unit-of-flow cost $c_{ij} \in \mathbb{R}_{>0}$, a fixed cost $f_{ij} \in \mathbb{R}_{>0}$, and a capacity $u_{ij} \in \mathbb{N}_{>0}$. Let $\mathcal{K}$ denote a set of commodities, each of which has a single source $o_k \in \mathcal{N}$ (also referred to as the commodity's origin), a single sink $d_k \in \mathcal{N}$ (also referred to as the commodity's destination), and a quantity q_k that must be routed along a single path from source to sink. Finally, let $e_k \in \mathbb{N}_{\geq 0}$ denote the time commodity k becomes available at its origin and $l_k \in \mathbb{N}_{\geq 0}$ denote the time it is due at its destination. Without loss of generality, we assume that $\min_{k \in \mathcal{K}} e_k = 0$. The Service Network Design Problem (SNDP) seeks to determine paths for the commodities and the resources required to transport the commodities along these paths so as to minimize the total cost, i.e., fixed and flow costs, and ensure that time constraints on the commodities are respected. The SNDP is typically modeled using a time-expanded network. A time-expanded network $\mathcal{D}_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{H}_\mathcal{T} \cup \mathcal{A}_\mathcal{T})$ is derived from $\mathcal{D}$ and a set of time points $\mathcal{T} = \bigcup_{i \in \mathcal{N}} \mathcal{T}_i$ with $\mathcal{T}_i = \{t^i_1, \ldots, t^i_{n_i}\}$. The node set $\mathcal{N}_\mathcal{T}$ has a node (i, t) for each $i \in \mathcal{N}$ and $t \in \mathcal{T}_i$. The arc set $\mathcal{H}_\mathcal{T}$ contains the arcs $((i, t^i_k), (i, t^i_{k+1}))$ for all $i \in \mathcal{N}$ and $k = 1, \ldots, n_i - 1$, known as holdover arcs, and the arc set $\mathcal{A}_\mathcal{T}$ contains arcs of the form $((i, t), (j, \bar{t}))$ where $(i, j) \in \mathcal{A}$, $t \in \mathcal{T}_i$, and $\bar{t} \in \mathcal{T}_j$. Note that $\mathcal{N}_\mathcal{T}$ uniquely determines $\mathcal{H}_\mathcal{T}$, and that, henceforth, we will, for any given $\mathcal{N}_\mathcal{T}$, make use of $\mathcal{H}_\mathcal{T}$ without explicit definition.

Arcs of the form $((i, t^i_k), (i, t^i_{k+1}))$ model the possibility of holding freight in location i, which may be advantageous if the freight can be consolidated with freight that arrives in location i at a later point in time. We assume that freight can be held at a location at no cost. The algorithm to be presented in the next section relies critically on this assumption. To achieve freight consolidation and be profitable, many consolidation carriers own and

operate their own network of terminals. For those carriers, holding freight at a terminal for a short amount of time, if necessary, incurs little or no additional costs and this assumption is appropriate and not limiting. Carriers that operate out of third-party-owned terminals may incur additional costs when holding freight. However, those costs are typically significantly less than the transportation savings achieved by holding freight to achieve consolidation, and thus not modeling them is unlikely to lead to the wrong decision.

Arcs of the form $((i,t),(j,\bar{t}))$ model the possibility to dispatch freight from location i at time t to arrive at location j at time $\bar{t}$. Note that an arc $((i,t),(j,\bar{t}))$ does *not* have to satisfy $\bar{t} - t = \tau_{ij}$. In fact, the flexibility to introduce arcs $((i,t),(j,\bar{t}))$ with a travel time that deviates from the actual travel time τ_{ij} of arc (i,j) is an essential feature of time-expanded networks and provides a mechanism to control the size of the time-expanded network. Unfortunately, deviating from the actual travel times also introduces approximations that may have undesirable effects. Consider, for example, using a discretization of time into hours and modeling travel from Chicago, IL to Milwaukee, WI, which takes about 95 minutes if departure is at 6 pm. When creating an arc $((\text{Chicago},18{:}00),(\text{Milwaukee},\bar{t}))$ one must choose whether to set $\bar{t}$ =19:00 or $\bar{t}$ =20:00. Both choices have downsides. Setting $\bar{t}$ =19:00 implies that a service network design model using this time-expanded network perceives freight traveling on this arc as arriving in Milwaukee in time to consolidate with freight departing from Milwaukee at 19:00, which is not actually possible. However, setting $\bar{t}$ =20:00 implies that a service network design model using this time-expanded network perceives freight destined for Milwaukee and due there at 19:45 traveling on this arc as arriving in Milwaukee too late, which is not the case. The latter shows that not only travel times have to be mapped onto the time-expanded network, but also the times that commodities are available at their origin and due at their destination. The typical mapping rounds up travel times, rounds up times that commodities are available, and rounds down times that commodities are due, because this ensures that any feasible solution to the SNDP model on the time-expanded network can be converted

to a true feasible solution, i.e., a feasible solution in real or continuous time.

A regular and fully time-expanded network $\mathcal{D}^{\Delta}_{\mathcal{T}}$ associated with $\mathcal{D}$ and discretization parameter $\Delta \in \mathbb{N}_{>0}$ has $\mathcal{T}_i = \{0, \Delta, 2\Delta, \ldots, K\Delta\}$ for all $i \in \mathcal{N}$ and for $K \in \mathbb{N}_{>0}$ with $\max_{k\in\mathcal{K}} l_k/\Delta \leq K < \max_{k\in\mathcal{K}} l_k/\Delta + 1$. Furthermore, for every arc $(i,j) \in \mathcal{A}$ and every node $(i,t) \in \mathcal{N}_{\mathcal{T}}$, there is an arc $((i,t),(j,t+\Delta\lceil\tau_{ij}/\Delta\rceil)$ in $\mathcal{A}_{\mathcal{T}}$ (unless $t+\Delta\lceil\tau_{ij}/\Delta\rceil) > K\Delta$). The networks $\mathcal{D}^{\Delta}_{\mathcal{T}}$ have become a popular tool in the design of approximation algorithms for flow-over-time problems, where they are known as condensed time-expanded networks (Fleischer and Skutella, [2007]; Groß et al., [2012]; Groß and Skutella, [2012]).

We define SND($\mathcal{D}_{\mathcal{T}}$) to be the service network design problem defined over a time-expanded network $\mathcal{D}_{\mathcal{T}}$. Let $y_{ij}^{t\bar{t}}$ denote the number of times arc (i,j) must be installed to accommodate dispatches from i at time t arriving at time $\bar{t}$ in j. (Because these variables capture resource movements, e.g., truck or trailer movements, we allow $y_{ij}^{t\bar{t}}$ to take on values greater than one.) Let $x_{ij}^{kt\bar{t}}$ represent whether commodity $k \in \mathcal{K}$ travels from i to j departing at time t to arrive at $\bar{t}$. Since we have assumed that a commodity must follow a single path from its origin to its destination the variables $x_{ij}^{kt\bar{t}}$ are binary. For presentational convenience, we assume that the nodes (o_k, e_k) and (d_k, l_k) are in $\mathcal{N}_{\mathcal{T}}$ for all $k \in \mathcal{K}$. (Otherwise, the nodes (o_k, t) with $t = \arg\min\{s \in \mathcal{T}_i \mid s > e_k\}$ and (d_k, t') with $t' = \arg\max\{s \in \mathcal{T}_i \mid s < l_k\}$ can be used instead.)

Thus, SND($\mathcal{D}_{\mathcal{T}}$) seeks

$$z(\mathcal{D}_{\mathcal{T}}) = \text{minimize} \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}} f_{ij} y_{ij}^{t\bar{t}} + \sum_{k\in\mathcal{K}} \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}} c_{ij} q_k x_{ij}^{kt\bar{t}}$$

subject to

$$\sum_{((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}\cup\mathcal{H}_{\mathcal{T}}} x_{ij}^{kt\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}_{\mathcal{T}}\cup\mathcal{H}_{\mathcal{T}}} x_{ji}^{k\bar{t}t} = \begin{cases} 1 & (i,t) = (o_k, e_k) \\ -1 & (i,t) = (d_k, l_k) \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in \mathcal{K}, (i,t) \in \mathcal{N}_{\mathcal{T}}, \tag{3.1}$$

$$\sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}} \leq u_{ij} y_{ij}^{t\bar{t}} \quad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}_{\mathcal{T}}, \tag{3.2}$$

$$x_{ij}^{kt\bar{t}} \in \{0,1\} \quad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}_{\mathcal{T}} \cup \mathcal{H}_{\mathcal{T}}, k \in \mathcal{K}, \tag{3.3}$$

$$y_{ij}^{t\bar{t}} \in \mathbb{N}_{\geq 0} \quad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}_{\mathcal{T}}. \tag{3.4}$$

That is, SND($\mathcal{D}_\mathcal{T}$) seeks to minimize the sum of fixed costs (the first term, which models transportation-related costs) and variable costs (the second term, which models handling-related costs). Note that we implicitly assume that holding freight at a location does not result in additional costs. Constraints (3.1) ensure that each commodity departs from its origin when it becomes available and arrives at its destination when it is due. Note the presence of holdover arcs allows a commodity to arrive early at its destination or depart late from its origin. Constraints (3.2) ensure that sufficient trailer capacity is available for the commodities that are sent from location i at time t to location j at time $\bar{t}$. Constraints (3.3) and (3.4) define the variables and their domains. We denote an optimal solution to this problem by $(x(\mathcal{D}_\mathcal{T}), y(\mathcal{D}_\mathcal{T}))$ and its value with $z(\mathcal{D}_\mathcal{T})$.

Observe that when using a regular and fully time-expanded network $\mathcal{D}^\Delta_\mathcal{T}$, no approximations are introduced when $\tau_{ij}/\Delta,\ e_k/\Delta$, and l_k/Δ are naturally integer. In that case, a feasible solution to SND($\mathcal{D}^\Delta_\mathcal{T}$) is also feasible in continuous time and an optimal solution to SND($\mathcal{D}^\Delta_\mathcal{T}$) is also optimal in continuous time. Let

$$\hat{\Delta} = GCD(GCD_{(i,j)\in\mathcal{A}}\tau_{ij},\ GCD_{k\in\mathcal{K}}e_k,\ GCD_{k\in\mathcal{K}}l_k),$$

where GCD is the greatest common divisor. We define CTSNDP to be SND($\mathcal{D}^{\hat{\Delta}}_\mathcal{T}$). We use $\hat{\mathcal{T}} = \bigcup_{i\in\mathcal{N}}\hat{\mathcal{T}}_i$ to denote the time points included in $\mathcal{D}^{\hat{\Delta}}_\mathcal{T}$, $\mathcal{N}^{\hat{\Delta}}_\mathcal{T}$ to denote its nodes, and $\mathcal{A}^{\hat{\Delta}}_\mathcal{T} \cup \mathcal{H}^{\hat{\Delta}}_\mathcal{T}$ to denote its arcs.

The fully time-expanded network $\mathcal{D}^{\hat{\Delta}}_\mathcal{T}$ tends to be prohibitively large for practical instances. Furthermore, it typically contains nodes that are superfluous. For example, a node $(i,t) \in \mathcal{N}^{\hat{\Delta}}_\mathcal{T}$ that no commodity $k \in \mathcal{K}$ can visit (possibly because doing so would prevent the commodity reaching its destination on time) is superfluous. Therefore, a fundamental question is whether a smaller of set of nodes $\mathcal{N}_\mathcal{T} \subset \mathcal{N}^{\hat{\Delta}}_\mathcal{T}$ and set of arcs $\mathcal{A}_\mathcal{T} \subset \mathcal{A}^{\hat{\Delta}}_\mathcal{T}$ can be determined *a priori*, such that solving SND($\mathcal{D}_\mathcal{T}$) yields an optimal solution to SND($\mathcal{D}^{\hat{\Delta}}_\mathcal{T}$). PROPOSITION 1 shows how to construct one such set $\mathcal{T}$.

Proposition 1. *To ensure that any optimal solution to SND($\mathcal{D}_\mathcal{T}$) is an optimal solution to CTSNDP, it is sufficient to include only time points in $\mathcal{T}$ that are determined by direct*

travel time paths starting at the origin of a commodity at the time that commodity becomes available, i.e., it is sufficient for $\mathcal{T}$ to consist only of time points of the form e_k for some commodity $k \in \mathcal{K}$, or of the form $e_k + \sum_{a \in P} \tau_a$ for some commodity $k \in \mathcal{K}$ and some path $P \subseteq \mathcal{A}$ originating at o_k.

Proof. Proof Consider an optimal (continuous time) solution. Shift all dispatch times to be as early as possible without changing any consolidations. This implies that each dispatch time at a node is now determined by the time a commodity originating at that node becomes available or by the arrival time of another commodity at the node. Suppose there is a dispatch time that is not at a time point of the form defined in the statement of the theorem. Choose the earliest such dispatch time t. Because this dispatch time t cannot occur at the time a commodity becomes available, it must be determined by the arrival time of a commodity, i.e., there must be a commodity dispatched on some arc $a \in \mathcal{A}$ at time $t' = t - \tau_a$. However, because of the choice of t and the assumption that all travel times are positive, it must be that t' is one of the time points defined in the statement of the theorem. But, since $t = t' + \tau_a$, t itself must be a time point of the form defined in the statement of the theorem, which contradicts its definition. □

The set of time points defined in PROPOSITION 1 may still be prohibitively large for practical instances, and is thus not enough, by itself, to enable solution of CTSNDP. However, it motivates, in part, one of the main ideas underlying our approach to solving CTSNDP. We iteratively refine (expand) a set of time points $\mathcal{T}$, containing time points 0, e_k and l_k for $k \in \mathcal{K}$, and some time points of the form $t = e_k + \sum_{a \in P} \tau_a$ for some commodity $k \in K$ and some path $P \subseteq \mathcal{A}$ originating at o_k, until we can prove that the solution to SND($\mathcal{D}_{\mathcal{T}}$) for a carefully chosen arc set $\mathcal{A}_{\mathcal{T}}$ can be converted to an optimal solution to CTSNDP. The details of the approach are provided in the next section.

3.4 An algorithm for solving CTSNDP

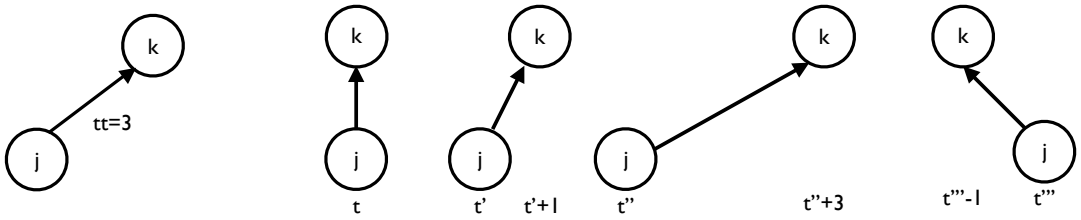
Our approach for solving CTSNDP can be thought of as a dual ascent procedure, because it repeatedly solves and refines a relaxation of CTSNDP until the solution to the relaxation can be converted to a feasible solution to CTSNDP of the same cost (and hence it is an optimal solution). Specifically, the approach repeatedly solves an instance of SND($\mathcal{D}_\mathcal{T}$) where $\mathcal{D}_\mathcal{T}$ has carefully chosen time points, carefully chosen arcs, and carefully chosen arc travel times. Because $\mathcal{T}_i$ may only contain a small subset of the time points in $\hat{\mathcal{T}}_i$ for $i \in \mathcal{N}$ and the set of time points at different locations may differ, i.e., $\mathcal{T}_i$ may be different from $\mathcal{T}_j$ for $i \neq j$, we refer to the time-expanded network $\mathcal{D}_\mathcal{T}$ as a *partially time-expanded* network. In the description of our algorithm, we will often refer to a "timed copy" of arc $(i, j) \in \mathcal{A}$ at node $(i, t) \in \mathcal{N}_\mathcal{T}$, which will mean an arc of the form $((i, t), (j, \bar{t})) \in \mathcal{A}_\mathcal{T}$. These partially time-expanded networks will have four important properties, which we discuss next. In all that follows, we will, for notational convenience, but without loss of generality, assume that $\hat{\Delta} = 1$.

PROPERTY 1. For all commodities $k \in \mathcal{K}$, the nodes (o_k, e_k) and (d_k, l_k) are in $\mathcal{N}_\mathcal{T}$.

PROPERTY 2. Every arc $((i, t), (j, \bar{t})) \in \mathcal{A}_\mathcal{T}$ has $\bar{t} \leq t + \tau_{ij}$.

PROPERTY 3. For every arc $a = (i, j) \in \mathcal{A}$ in the flat network, $\mathcal{D}$, and for every node (i, t) in the partially time-expanded network, $\mathcal{D}_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{A}_\mathcal{T} \cup \mathcal{H}_\mathcal{T})$, there is a timed copy of a in $\mathcal{A}_\mathcal{T}$ starting at (i, t).

PROPERTY 2 implies that we work with timed copies of an arc that are either of the correct length or are too short. This is illustrated in Figure 3.3, where we depict different timed copies of an arc $(j, k) \in \mathcal{A}$ that may be created by the algorithm. Observe that the lengths (or travel times) of the timed copies are different for the different dispatch times t, t', t'', and t''', and that the travel time of a timed copy may even be negative (as is the case for dispatch time t''').



(a) Flat arc (j, k). (b) Different timed copies of (j, k) can model travel time differently.

Figure 3.3: Travel times of timed copies of (j, k); travel times do not exceed the travel time of arc (j, k).

Definition 1. *If $\mathcal{D}_\mathcal{T}$ satisfies* PROPERTIES *2 and 3, we say that $\mathcal{D}_\mathcal{T}$ has the early-arrival property.*

In what follows, it is useful to observe that any sequence of (non-holdover) arcs,

$$((i_1, t_1), (j_1, t'_1)), ((i_2, t_2), (j_2, t'_2)), \ldots, ((i_\eta, t_\eta), (j_\eta, t'_\eta)),$$

in a time-expanded network, $\mathcal{D}_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{A}_\mathcal{T} \cup \mathcal{H}_\mathcal{T})$, where $((i_h, t_h), (j_h, t'_h)) \in \mathcal{A}_\mathcal{T}$ for each $h = 1, \ldots, \eta$, induces a (unique) valid path in $\mathcal{D}_\mathcal{T}$, formed by the addition of appropriate holdover arcs, provided that $j_h = i_{h+1}$ and $t'_h \leq t_{h+1}$, for all $h = 1, \ldots, \eta - 1$.

Definition 2. *For any pair of nodes in the flat network, $j, j' \in \mathcal{N}$, we define the* distance *from j to j', denoted by $\underline{\tau}_{j,j'}$, to be the length of any shortest path, in the flat network, from $j \in \mathcal{N}$ to $j' \in \mathcal{N}$, with respect to the travel times, τ.*

Lemma 1. *Let $\mathcal{D}_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{A}_\mathcal{T})$ be a partially time-expanded network that satisfies* PROPERTY *1 and has the early-arrival property. Then for each commodity $k \in \mathcal{K}$ and each node in the flat network, $i \in \mathcal{N}$, there exists a timed node $(i, t) \in \mathcal{N}_\mathcal{T}$, with $t \leq e_k + \underline{\tau}_{o_k,i}$.*

Proof. Proof Let $k \in \mathcal{K}$. The result holds trivially, by PROPERTY 1, if $i = o_k$, so consider $i \in \mathcal{N}$ with $i \neq o_k$. We proceed by induction on the number of arcs in the shortest path in the flat network from o_k to i with respect to τ.

Suppose $i \in \mathcal{N}$ has a shortest path from o_k given by the single arc $(o_k, i) \in \mathcal{A}$. Then it must be that $\underline{\tau}_{o_k,i} = \tau_{o_k,i}$. By PROPERTIES 1 – 3, there exists a $t \leq e_k + \tau_{o_k,i} = e_k + \underline{\tau}_{o_k,i}$ with $((o_k, e_k), (i, t)) \in \mathcal{A}_\mathcal{T}$, so it must be that $(i, t) \in \mathcal{N}_\mathcal{T}$, as required.

Now suppose that for all $i \in \mathcal{N}$ with a shortest path from o_k having $\eta - 1$ arcs, with $\eta \geq 2$, there exists timed node $(i,t) \in \mathcal{N}_\mathcal{T}$, with $t \leq e_k + \underline{\tau}_{o_k,i}$, and consider a node $j \in \mathcal{N}$ with a shortest path from o_k) having η arcs, say $P = ((j_0,j_1),(j_1,j_2),\ldots,(j_{\eta-1},j_\eta)$, where $j_0 = o_k$ and $j_\eta = j$ is such a path. By well-known properties of shortest paths, $((j_0,j_1),(j_1,j_2),\ldots,(j_{\eta-2},j_{\eta-1})$ is a shortest path from $j_0 = o_k$ to $j_{\eta-1}$, so, by the inductive assumption, there exists $(j_{\eta-1},t) \in \mathcal{N}_\mathcal{T}$, with $t \leq e_k + \underline{\tau}_{o_k,j_{\eta-1}}$. By PROPERTY 3, there must exist t' with $((j_{\eta-1},t),(j_\eta,t')) \in \mathcal{A}_\mathcal{T}$, and by PROPERTY 2, it must be that $t' \leq t + \tau_{j_{\eta-1},j_\eta}$ and hence

$$t' \leq e_k + \underline{\tau}_{o_k,j_{\eta-1}} + \tau_{j_{\eta-1},j_\eta} = e_k + \sum_{h=1}^{\eta-1} \tau_{j_{h-1},j_h} + \tau_{j_{\eta-1},j_\eta} = e_k + \sum_{h=1}^{\eta} \tau_{j_{h-1},j_h} = e_k + \underline{\tau}_{o_k,j_\eta}.$$

The result follows by induction.

□

Theorem 1. *Let $\mathcal{D}_\mathcal{T}$ be a partially time-expanded network that satisfies* PROPERTY *1 and has the early-arrival property. Then SND($\mathcal{D}_\mathcal{T}$) is a relaxation of the CTSNDP.*

Proof. Proof Consider an optimal solution $(\overline{x}(\mathcal{D}_\mathcal{T}^{\hat{\Delta}}), \overline{y}(\mathcal{D}_\mathcal{T}^{\hat{\Delta}}))$ to CTSNDP and let $\mathcal{A}^* = \{((i,t),(j,t+\tau_{ij})) \in \mathcal{A}_\mathcal{T}^{\hat{\Delta}} \mid \overline{y}_{ij}^{t,t+\tau_{ij}} > 0\}$ (recalling the assumption that $\hat{\Delta} = 1$). Furthermore, let $\mathcal{K}_{((i,t),(j,t+\tau_{ij}))}$ represent the set of commodities dispatched on arc $((i,t),(j,t+\tau_{ij})) \in A^*$, in this optimal solution, i.e., let

$$\mathcal{K}_{((i,t),(j,t+\tau_{ij}))} = \{k \in \mathcal{K} \mid \overline{x}_{ij}^{k,t,t+\tau_{ij}} > 0\}.$$

In what follows, we will identify each arc $a \in \mathcal{A}^*$ with a unique arc in $\mu(a) \in \mathcal{A}_\mathcal{T}$, and construct $(x(\mathcal{D}_\mathcal{T}), y(\mathcal{D}_\mathcal{T}))$ so that the commodity flow represented by x and trailer capacity represented by y, on each timed arc of the form $\mu(a)$, is exactly that of $\overline{x}$ and $\overline{y}$, respectively, on a, and so that (x,y) is feasible for SND($\mathcal{D}_\mathcal{T}$) and has cost identical to the optimal value of CTSNDP.

Specifically, we define the mapping $\mu : A^* \to \mathcal{A}_\mathcal{T}$ as follows. For a given $a = ((i,t),(j,t+\tau_{ij})) \in A^*$, $\mathcal{K}_a \neq \emptyset$, and for all $k \in \mathcal{K}_a$, it must be that $e_k + \underline{\tau}_{o_k,i} \leq t$. Thus, by Lemma 1, there exists $t' \leq t$ with $(i,t') \in \mathcal{N}_\mathcal{T}$. Therefore $\rho_i(t)$, defined to be the

latest time point at or before t so that $(i, \rho_i(t)) \in \mathcal{N}_\mathcal{T}$, i.e., $\rho_i(t) = \arg\max\{s \in \mathcal{T}_i \mid s \leq t\}$, is well defined. By PROPERTY 3, there must exists t' with $((i, \rho_i(t)), (j, t')) \in \mathcal{A}_\mathcal{T}$. Choose $\sigma(a)$ to be any such t', and define $\mu(a) = ((i, \rho_i(t)), (j, \sigma(a)))$.

We may now define the trailer capacity on each timed arc $\tilde{a} = ((i, \tilde{t}), (j, \tilde{t}')) \in \mathcal{A}_\mathcal{T}$ by summing the trailer capacities in the optimal solution on all arcs that map to $\tilde{a}$ under μ:

$$y_{ij}^{\tilde{t}\tilde{t}'} = \sum_{\substack{a=((i,t),(j,t+\tau_{ij}))\in A^* : \\ \mu(a)=\tilde{a}}} y_{ij}^{t,t+\tau_{ij}},$$

where the right-hand side is taken to be zero if no arcs in A^* map to $\tilde{a}$, under μ.

To construct the commodity flows in $\mathcal{D}_\mathcal{T}$, we will show that the sequence of (non-holdover) arcs, in each commodity's path in the optimal solution, maps, under μ, to a sequence of arcs in $\mathcal{A}_\mathcal{T}$ that induce a valid path in $\mathcal{D}_\mathcal{T}$ from the commodity's origin node in $\mathcal{N}_\mathcal{T}$ to its destination in $\mathcal{N}_\mathcal{T}$.

For each commodity k, let $\overline{P}_\mathcal{T}^k$ denote the path from (o_k, e_k) to (d_k, l_k) in $\mathcal{D}_\mathcal{T}^{\hat{\Delta}}$ induced by the optimal commodity flow, $\overline{x}$, i.e.,

$$\overline{P}_\mathcal{T}^k = \{((i,t),(j,t')) \in \mathcal{A}_\mathcal{T}^{\hat{\Delta}} \cup \mathcal{H}_\mathcal{T}^{\hat{\Delta}} \mid \overline{x}_{ij}^{k,t,t'} > 0\} = \{((i,t),(j,t')) \in A^* \cup \mathcal{H}_\mathcal{T}^{\hat{\Delta}} \mid \overline{x}_{ij}^{k,t,t'} = 1\}.$$

Say $\overline{P}_\mathcal{T}^k$ is the path uniquely induced by the sequence of arcs $a^1, \ldots, a^\eta \in A^*$, together with holdover arcs linking from (o_k, e_k) to the head of a^1 and from the tail of a^η to (d_k, l_k). Then we may write $a^h = ((i_h, t_h), (i_{h+1}, t_{h+1}))$, with $t_{h+1} = t_h + \tau_{ij}$, for $h = 1, \ldots, \eta - 1$.

We claim that the sequence of arcs $\mu(a^1), \ldots, \mu(a^\eta) \in \mathcal{A}_\mathcal{T}$ induces, with the addition of appropriate holdover arcs, a valid path in $\mathcal{D}_\mathcal{T}$ from (o_k, e_k) to (d_k, l_k). To prove this claim, observe that for any $h \in \{1, \ldots, \eta - 1\}$, we have $\mu(a^h) = ((i_h, \rho_{i_h}(t_h)), (i_{h+1}, \sigma(a^h)))$, where $\rho_{i_h}(t_h) = \arg\max\{s \in \mathcal{T}_{i_h} \mid s \leq t_h\}$ and so $\rho_{i_h}(t_h) \leq t_h$, and, by PROPERTY 2, $\sigma(a^h) \leq \rho_{i_h}(t_h) + \tau_{ij}$. Thus $\sigma(a^h) \leq t_h + \tau_{ij} = t_{h+1}$. Recall that, by definition, $\sigma(a^h) \in \mathcal{T}_{i_{h+1}}$ and so it must be that $\rho_{i_{h+1}}(t_{h+1}) \geq \sigma(a^h)$. This shows that $\mu(a^1), \ldots, \mu(a^\eta)$ induces a valid path in $\mathcal{D}_\mathcal{T}$ from $(o_k, \rho_{o_k}(t_1))$ to $(d_k, \sigma(a^\eta))$. Now it must be that $\rho_{o_k}(t_1) \geq e_k$, since $t_1 \geq e_k$, and it must be that $\sigma(a^\eta) \leq l_k$ since $\sigma(a^\eta) \leq t_{\eta+1} \leq l_k$. Thus by including appropriate holdover arcs at o_k and d_k, our claim follows, and we set $x_{ij}^{k,t,t'} = 1$ for all arcs

$((i,t),(j,t')) \in \mathcal{A}_\mathcal{T} \cup \mathcal{H}_\mathcal{T}$ that are in the resulting path in $\mathcal{D}_\mathcal{T}$ from (o_k, e_k) to (d_k, l_k).

Now, it is not hard to see that solution (x, y) constructed in this way is feasible for SND($\mathcal{D}_\mathcal{T}$), and replicates solution $(\overline{x}, \overline{y})$ to CTSNDP in the sense that the commodities flow along the same paths (in the flat network) and the same consolidations occur. Hence the two solutions have identical objective function value. Thus SND($\mathcal{D}_\mathcal{T}$) is a relaxation of the CTSNDP. □

Note that the proof, and thus the theorem, relies on the fact that holding freight at a location does not result in additional cost.

The following lemma regarding partially time-expanded networks with the early-arrival property will be useful when we refine a partially time-expanded network during the course of our algorithm. We omit its proof, since it follows immediately from the definitions of PROPERTIES 2 and 3 .

Lemma 2. *If a partially time-expanded network $\mathcal{D}_\mathcal{T}$ has the early-arrival property, $((i,t),(j,t')) \in \mathcal{A}_\mathcal{T}$, and $(j,t'') \in \mathcal{N}_\mathcal{T}$ with $t'' \leq t + \tau_{ij}$, then the partially time-expanded network in which arc $((i,t),(j,t'))$ is replaced with arc $((i,t),(j,t''))$ will also have the early-arrival property.*

There are many partially time-expanded networks $\mathcal{D}_\mathcal{T}$ that satisfy PROPERTIES 1, 2, and 3. We restrict ourselves to partially time-expanded networks with arc sets $\mathcal{A}_\mathcal{T}$ that satisfy one additional property.

PROPERTY 4. If arc $((i,t),(j,t')) \in \mathcal{A}_\mathcal{T}$, then there does not exist a node $(j,t'') \in \mathcal{N}_\mathcal{T}$ with $t' < t'' \leq t + \tau_{ij}$.

Definition 3. *If $\mathcal{D}_\mathcal{T}$ satisfies* PROPERTY *4, we say that $\mathcal{D}_\mathcal{T}$ has the longest-feasible-arc property.*

Observe that, for a given $\mathcal{T}$, (and $\mathcal{N}_\mathcal{T}$), there is a unique set of timed arcs that satisfy both the early-arrival and the longest-feasible-arc properties. To see this, first note if $((i,t),(j,t'))$ and $((i,t),(j,t''))$ are both in $\mathcal{A}_\mathcal{T}$ for some $t' \neq t''$, where without loss of

generality $t' < t''$, then $t' < t'' \leq t + \tau_{ij}$ by PROPERTY 2, and the longest-feasible-arc property fails. Thus for each $(i,t) \in \mathcal{N}_\mathcal{T}$ and each $(i,j) \in \mathcal{A}$, there can be at most one arc of the form $((i,t),(j,t')$ in $\mathcal{A}_\mathcal{T}$ satisfying both properties. For $\mathcal{A}_\mathcal{T}$ satisfying PROPERTY 3 there must be at at least one such arc. Hence, if $\mathcal{A}_\mathcal{T}$ satisfies both properties, there is exactly one arc of the form $((i,t),(j,t')$ in $\mathcal{A}_\mathcal{T}$ for each $(i,t) \in \mathcal{N}_\mathcal{T}$ and $(i,j) \in \mathcal{A}$. By PROPERTY 2 and PROPERTY 4, it must be that $t' = \arg\max\{s \mid s \leq t + \tau_{ij},\ (j,s) \in \mathcal{N}_\mathcal{T}\}$.

The reason for restricting ourselves to arc sets with the longest-feasible-arc property is the following theorem.

Theorem 2. *For a fixed $\mathcal{T}$, (and $\mathcal{N}_\mathcal{T}$), among the partially time-expanded networks $\mathcal{D}_\mathcal{T}$ with the early-arrival property, the one with the longest-feasible arc property induces an instance of SND($\mathcal{D}_\mathcal{T}$) with the largest optimal objective function value.*

Proof. Proof Consider a partially time-expanded network $\mathcal{D}^{LF}_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{A}^{LF}_\mathcal{T} \cup \mathcal{H}_\mathcal{T})$ with arc set $\mathcal{A}^{LF}_\mathcal{T}$ that has the longest-feasible-arc property and a partially time-expanded network $\mathcal{D}'_\mathcal{T} = (\mathcal{N}_\mathcal{T}, \mathcal{A}'_\mathcal{T} \cup \mathcal{H}_\mathcal{T})$ with arc set $\mathcal{A}'_\mathcal{T}$ that does not. Assume that both networks have the early-arrival property. We will show that any solution to SND($\mathcal{D}^{LF}_\mathcal{T}$) can be converted to a solution to SND($\mathcal{D}'_\mathcal{T}$) of equal value. Thus, the optimal objective function value of SND($\mathcal{D}'_\mathcal{T}$) can be no greater than that of SND($\mathcal{D}^{LF}_\mathcal{T}$).

Consider a solution $(x(\mathcal{D}^{LF}_\mathcal{T}), y(\mathcal{D}^{LF}_\mathcal{T}))$ to the SND($\mathcal{D}^{LF}_\mathcal{T}$) and an arc $((i,t),(j,t')) \in \mathcal{A}^{LF}_\mathcal{T}$ such that $y^{tt'}_{ij}(\mathcal{D}^{LF}_\mathcal{T}) > 0$ and all arcs of the form $((i,t),(j,t'')) \in \mathcal{A}'_\mathcal{T}$ have $t'' < t'$. If no such arc exists, then the solution is clearly feasible for SND($\mathcal{D}'_\mathcal{T}$). Thus, suppose such an arc exists.

Because both networks are defined on the same node set $\mathcal{N}_\mathcal{T}$, the path from (j,t'') to (j,t') exists in $(\mathcal{N}_\mathcal{T}, \mathcal{H}_\mathcal{T})$. Consequently, we can adapt the solution $(x(\mathcal{D}^{LF}_\mathcal{T}), y(\mathcal{D}^{LF}_\mathcal{T}))$ for this arc to one for the SND($\mathcal{D}'_\mathcal{T}$) by assigning $y^{tt''}_{ij}(\mathcal{D}'_\mathcal{T}) = y^{tt'}_{ij}(\mathcal{D}^{LF}_\mathcal{T})$, and routing the corresponding commodity flows on the path formed by concatenating arc $((i,t),(j,t''))$ with the path from (j,t'') to (j,t'). Note that the cost of this change is 0, because we have assumed that there is no cost associated with using the holdover arcs. Because this change

leaves any commodities that traveled on the arc $((i,t),(j,t'))$ in the solution to SND($\mathcal{D}_{\mathcal{T}}^{LF}$) at the same node (j,t'), we can repeat this process one arc at a time and are left with a solution to the SND($\mathcal{D}_{\mathcal{T}}^{'}$) of equal value. □

THEOREM 1, and to a lesser extent THEOREM 2, provide the basis for our iterative-refinement algorithm for solving CTNSNDP; Algorithm 1 presents a high-level overview.

Algorithm 1 SOLVE-CTSNDP

Require: Flat network $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, commodity set $\mathcal{K}$

Create a partially time-expanded network $\mathcal{D}_{\mathcal{T}}$ satisfying Properties 1, 2, 3, and 4
while not solved **do**
    Solve SND($\mathcal{D}_{\mathcal{T}}$)
    Determine whether the solution to SND($\mathcal{D}_{\mathcal{T}}$) can be converted to a feasible solution to CTSNDP with the same cost
    if it can be converted **then**
        Stop. The converted solution is optimal for CTSNDP.
    else
        The solution to SND($\mathcal{D}_{\mathcal{T}}$) must use at least one arc that is "too short". Refine the partially time-expanded network $\mathcal{D}_{\mathcal{T}}$ by correcting the length of at least one such arc, in the process adding at least one new time point to $\mathcal{T}_i$ for some $i \in \mathcal{N}$.

Before discussing the various components of the algorithm in more detail, we prove the following result.

Theorem 3. SOLVE-CTSNDP *terminates with an optimal solution.*

Proof. Proof The algorithm terminates when the optimal solution to SND($\mathcal{D}_{\mathcal{T}}$) can be converted to a solution of CTSNDP with the same cost. Because SND($\mathcal{D}_{\mathcal{T}}$) is a relaxation of CTSNDP (THEOREM 1), the converted solution must be an optimal solution to CTSNDP.

Furthermore, at every iteration in which SOLVE-CTSNDP does not terminate, the length of at least one arc $a \in \mathcal{A}_{\mathcal{T}}$ is increased to its correct length. Because there are a finite number of time points and arcs, at some iteration all arcs in $\mathcal{A}_{\mathcal{T}}$ must have travel times that correspond to the actual travel time of the corresponding arc in the flat network, in which case the solution to SND($\mathcal{D}_{\mathcal{T}}$) is a solution to CTSNDP and the algorithm terminates. □

Because arcs in $\mathcal{A}_\mathcal{T}$ can be too short, it is possible that a solution to SND($\mathcal{D}_\mathcal{T}$) contains a path for a commodity $k \in \mathcal{K}$ that is too long, i.e., its actual length or duration exceeds the available time $l_k - e_k$. We avoid such solutions by adding valid inequalities to SND($\mathcal{D}_\mathcal{T}$). More specifically, we prevent paths that are too long by adding the following inequality to SND($\mathcal{D}_\mathcal{T}$):

$$\sum_{((i,t),(j,t'))\in\mathcal{A}_\mathcal{T}} \tau_{ij} x_{ij}^{k\tau'} \leq l_k - e_k. \tag{3.5}$$

3.4.1 Creating an initial partially time-expanded network

The initial partially time-expanded network consists of nodes (o_k, e_k) and (d_k, l_k) for all $k \in \mathcal{K}$ and $(u, 0)$ for all $u \in \mathcal{N}$. For each node $(i, t) \in \mathcal{N}_\mathcal{T}$ and arc $(i, j) \in \mathcal{A}$, we find the node (j, t') with largest t' such that $t' \leq t + \tau_{ij}$ and add arc $((i, t), (j, t'))$ to $\mathcal{A}_\mathcal{T}$. Note that because $\mathcal{N}_\mathcal{T}$ includes nodes $(u, 0)$ for $u \in \mathcal{N}$, it is always possible to find such a node (j, t'). (Note, too, that we may have $t' < t$, in which case the arc travels backward in time.) Finally, for all nodes (i, t) and (i, t') such that t' is the smallest time point with $t' > t$, we add arc $((i, t), (i, t'))$ to $\mathcal{H}_\mathcal{T}$. It is not hard to see that this partially time-expanded network satisfies PROPERTIES 1, 2, 3, and 4. For a detailed description of CREATE-INITIAL, see Algorithm 2.

Algorithm 2 CREATE-INITIAL

Require: Directed network $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, commodity set $\mathcal{K}$

1: **for all** $k \in \mathcal{K}$ **do**
2: Add node (o_k, e_k) to $\mathcal{N}_\mathcal{T}$
3: Add node (d_k, l_k) to $\mathcal{N}_\mathcal{T}$
4: **for all** $u \in \mathcal{N}$ **do**
5: Add node $(u, 0)$ to $\mathcal{N}_\mathcal{T}$
6: **for all** $(i, t) \in \mathcal{N}_\mathcal{T}$ **do**
7: **for all** $(i, j) \in \mathcal{A}$ **do**
8: Find largest $t^{'}$ such that $(j, t') \in \mathcal{N}_\mathcal{T}$ and $t^{'} \leq t + \tau_{ij}$ and add arc $((i, t), (j, t'))$ to $\mathcal{A}_\mathcal{T}$
9: Find smallest t' such that $(i, t') \in \mathcal{N}_\mathcal{T}$ and $t' > t$ and add arc $((i, t), (i, t'))$ to $\mathcal{H}_\mathcal{T}$

3.4.2 Refining a partially time-expanded network

It is necessary to refine $\mathcal{D}_\mathcal{T}$ when the solution to SND($\mathcal{D}_\mathcal{T}$) cannot be converted to a feasible solution to CTSNDP with the same cost, which can happen when an arc in $\mathcal{A}_\mathcal{T}$ is "too short." When refining $\mathcal{D}_\mathcal{T}$, we ensure that (1) the length of at least one arc that is too short is corrected, and (2) that the resulting partially time-expanded network again satisfies PROPERTIES 1, 2, 3, and 4.

More specifically, when we lengthen an arc $((i,t),(j,t'))$ that is too short, i.e., $t' < t+\tau_{ij}$, we replace it with the arc $((i,t),(j,t+\tau_{ij}))$. Because $\mathcal{D}_\mathcal{T}$ has the longest-feasible-arc property, node $(j,t+\tau_{ij})$ was not in $\mathcal{N}_\mathcal{T}$ and will have to be added to $\mathcal{N}_\mathcal{T}$. Lengthening arc $((i,t),(j,t'))$ to $((i,t),(j,t+t_{ij}))$ is a 2-step process based on the following two lemmas. Details of the two steps are provided in Algorithm 4 and Algorithm 5, respectively, and applied in sequence in Algorithm 3.

Lemma 3. *If a time-expanded network $\mathcal{D}_\mathcal{T}$ has the early-arrival property, and (1) a new time point t^i_{new} with $t^i_k < t^i_{new} < t^i_{k+1}$ is added to $\mathcal{T}_i = \{t^i_1,\ldots,t^i_{n_i}\}$, (2) a new node (i,t^i_{new}) is added to $\mathcal{N}_\mathcal{T}$, and (3) for every arc $((i,t^i_k)(j,\bar{t}))$ in $\mathcal{D}_\mathcal{T}$, a new arc $((i,t^i_{new}),(j,\bar{t}))$ is added to $\mathcal{A}_\mathcal{T}$, then the resulting time-expanded network again has the early-arrival property.*

Proof. Proof The only new arcs added to $\mathcal{A}_\mathcal{T}$ are those of the form $((i,t^i_{new}),(j,\bar{t}))$ where $((i,t^i_k),(j,\bar{t}))$ was already in $\mathcal{A}_\mathcal{T}$ and $t^i_{new} > t^i_k$. If $\mathcal{D}_\mathcal{T}$ already satisfied Property 2, then $\bar{t} \le t^i_k + \tau_{ij}$. Hence $\bar{t} < t^i_{new} + \tau_{ij}$, and Property 2 is preserved. The only new node added is (i,t^i_{new}). Now if $\mathcal{D}_\mathcal{T}$ already satisfied Property 3, it must be that for all $(i,j) \in \mathcal{A}$, there exists a timed arc $((i,t^i_k),(j,\bar{t})) \in \mathcal{A}_\mathcal{T}$ for some $\bar{t}$. But for each such arc, the new arc $((i,t^i_{new}),(j,\bar{t}))$ is added to $\mathcal{A}_\mathcal{T}$. Thus Property 3 is preserved, too. □

Unfortunately, after adding the new time point, the new node, and the new arcs, the partially time-expanded network no longer satisfies the longest-feasible-arc property.

However, a few simple changes to the network restore the longest-feasible-arc property while maintaining the early-arrival property, as is shown in the following lemma.

Lemma 4. *After refining a partially time-expanded network $\mathcal{D}_\mathcal{T}$ having the longest-feasible-arc property by adding new time point t^i_{new} with $t^i_k < t^i_{new} < t^i_{k+1}$ to $\mathcal{T}_i$, adding new node (i, t^i_{new}) to $\mathcal{N}_\mathcal{T}$, and adding for every arc $((i, t^i_k)(j, \bar{t}))$ in $\mathcal{D}_\mathcal{T}$, a new arc $((i, t^i_{new}), (j, \bar{t}))$ to $\mathcal{A}_\mathcal{T}$, the longest-feasible-arc property will be restored by*

1. *replacing every arc $((j, t'), (i, t^i_k))$ with $t' + \tau_{ji} \geq t^i_{new}$ with arc $((j, t'), (i, t^i_{new}))$, and*
2. *finding, for every new arc, $((i, t^i_{new}), (j, \bar{t}))$, the node, (i, t'), with largest t', such that $\bar{t} < t' \leq t^i_{new} + \tau_{ij}$, and, if such a node exists, replacing arc $((i, t^i_{new}), (j, \bar{t}))$ with arc $((i, t^i_{new}), (j, t'))$.*

Proof. Proof The only arcs in $\mathcal{D}_\mathcal{T}$ that may violate the longest-feasible-arc property after the introduction of the new node (i, t^i_{new}) are those with head (i, t^i_k). These arcs are replaced if needed. The newly added arcs may also violate the longest-feasible-arc property, but are replaced if needed. □

When Algorithm 4, REFINE, is applied to a partially time-expanded network with the early-arrival property, Lemma 3 ensures that the resulting partially time-expanded network will also have the early-arrival property. When Algorithm 5, RESTORE, is applied to a partially time-expanded network with the early-arrival property, Lemma 2 guarantees the property is maintained. Lemma 4 ensures both steps preserve the longest-feasible-arc property. Thus Algorithm 3, LENGTHEN-ARC, preserves both the early-arrival and longest-feasible-arc properties.

Algorithm 3 LENGTHEN-ARC$((i, t), (j, t'))$

Require: Arc $((i, t), (j, t')) \in \mathcal{A}_\mathcal{T}$
1: REFINE$(j, t + \tau_{ij})$
2: RESTORE$(j, t + \tau_{ij})$

Algorithm 4 REFINE(i, t^i_{new})

Require: Node $i \in \mathcal{N}$; time point $t^i_{new} \in \mathcal{T}_i$ with $t^i_k < t^i_{new} < t^i_{k+1}$

1: Add node (i, t^i_{new}) to $\mathcal{N}_\mathcal{T}$;
2: Delete arc $((i, t^i_k), (i, t^i_{k+1}))$ from $\mathcal{A}_\mathcal{T}$; add arcs $((i, t^i_k), (i, t^i_{new}))$ and $((i, t^i_{new}), (i, t^i_{k+1}))$ to $\mathcal{A}_\mathcal{T}$
3: **for** $((i, t^i_k), (j, t)) \in \mathcal{A}_\mathcal{T}$ **do**
4: Add arc $((i, t^i_{new}), (j, t))$ to $\mathcal{A}_\mathcal{T}$

Algorithm 5 RESTORE(i, t^i_{new})

Require: Node $i \in \mathcal{N}$; time point $t^i_{new} \in \mathcal{T}_i$ with $t^i_k < t^i_{new} < t^i_{k+1}$

1: **for all** $((i, t^i_k), (j, t)) \in \mathcal{A}_\mathcal{T}$ **do**
2: Set $t' = \arg\max\{s \in \mathcal{T}_j \mid s \leq t^i_{new} + \tau_{ij}\}$.
3: **if** $t' \neq t$ **then**
4: Delete arc $((i, t^i_{new}), (j, t))$ from $\mathcal{A}_\mathcal{T}$; add arc $((i, t^i_{new}), (j, t'))$ to $\mathcal{A}_\mathcal{T}$
5: **for all** $((j, t), (i, t^i_k)) \in \mathcal{A}_\mathcal{T}$ such that $t + \tau_{ji} \geq t^i_{new}$ **do**
6: Delete arc $((j, t), (i, t^i_k))$ from $\mathcal{A}_\mathcal{T}$; add arc $((j, t), (i, t^i_{new}))$ to $\mathcal{A}_\mathcal{T}$

OBSERVATION 1. Because LENGTHEN-ARC takes a timed arc that is too short and replaces it with a timed arc that has the correct length, i.e., the actual travel time, the length of an arc is corrected at most once. This implies that SOLVE-CTSNDP, with LENGTHEN-ARC used in step 8, will terminate in a finite number of iterations. In particular, the number of iterations is bounded above by $|\hat{\mathcal{T}}||\mathcal{A}|$, since this is an upper bound on the number of timed arcs.

The reason for refining the partially time-expanded network is that the solution $(x(\mathcal{D}_\mathcal{T}), y(\mathcal{D}_\mathcal{T}))$ to SND($\mathcal{D}_\mathcal{T}$) cannot be converted to a solution to CTSNDP with equal value. A solution $(x(\mathcal{D}_\mathcal{T}), y(\mathcal{D}_\mathcal{T}))$ specifies the path each commodity k takes from its origin to its destination as well as the consolidations of commodities on arcs in the network, where a consolidation of commodities on an arc $(i, j) \in \mathcal{A}$ occurs when two or more commodities travel on that arc at the same time, i.e., $|\mathcal{K}_{((i,t),(j,t'))}| \geq 2$, where

$$\mathcal{K}_{((i,t),(j,t'))} = \{k \in \mathcal{K} \mid x^{k,t,t'}_{ij} = 1\}.$$

(Assuming $\sum_{k \in \mathcal{K}_{((i,t),(j,t'))}} q_k \leq u_{ij}$, these commodities will share the same resource, i.e., will be loaded into the same trailer, and the fixed cost f_{ij} is incurred only once.)

Because Constraints (3.5) ensure that the paths specified in the solution for the commodities are time-feasible with actual travel times, the solution cannot be converted to a solution to CTSNDP with equal value because the consolidations specified in the solution cannot be realized when the actual travel times are observed. This implies that there has to be a commodity $k \in \mathcal{K}$ that flows on an arc that is too short, i.e., there has to be a commodity k and an arc $((i,t),(j,t')$ with $t' < t + \tau_{ij}$ for which $x_{ij}^{k\tau'} = 1$. Next, we discuss how to identify arcs that are too short.

3.4.3 Identifying arcs to lengthen

We formulate the problem of identifying arcs to lengthen as a mixed integer program (MIP). For each $a \in \mathcal{A}_\mathcal{T}$, let J_a be the set of all pairs of commodities dispatched on a in the optimal solution to SND($\mathcal{D}_\mathcal{T}$), i.e., $J_a = \{(k_1, k_2) \in \mathcal{K}_a \times \mathcal{K}_a \mid k_1 < k_2\}$. Furthermore, let $\overline{\mathcal{J}}$ denote the set of arcs $a \in \mathcal{A}_\mathcal{T}$ on which more than one commodity is dispatched, i.e., $\overline{\mathcal{J}} = \{a \in \mathcal{A}_\mathcal{T} : |\mathcal{K}_a| \geqslant 2\}$. To formulate the MIP, for each commodity $k \in \mathcal{K}$, let $P_k = \{i_1^k = o_k, i_2^k, i_3^k, \ldots, i_p^k = d_k\}$ represent the path that commodity k follows from its source to its sink, in terms of the nodes it visits along the way, in the optimal solution to SND($\mathcal{D}_\mathcal{T}$), and, for each arc $(i_j, i_{j+1}) \in P_k$, let $\overline{\tau}^k_{i_j i_{j+1}}$ represent the travel time modeled in $\mathcal{D}_\mathcal{T}$ for that arc. Then, for each $k \in \mathcal{K}$ and each node i_j^k in path P_k, define variables $\gamma_{i_j}^k \geq 0$ to represent the dispatch time of commodity k at node i_j^k, and for each $a \in \mathcal{A}_\mathcal{T}$ and each $k \in \mathcal{K}$, define two sets of variables: $\theta^k_{i_j i_{j+1}}$ to represent the travel time of arc (i_j, i_{j+1}) when taken by commodity k, and, $\sigma^k_{i_1 i_{j+1}}$ to represent whether the arc is allowed to be too short when taken by commodity k. With these variables, we define the following MIP to determine the fewest number of arcs that must be too short for the consolidations that occur in a solution to SND($\mathcal{D}_\mathcal{T}$) to be realized:

$$Z = \text{minimize} \sum_{k \in \mathcal{K}} \sum_{j=1}^{|P_k|-1} \sigma^k_{i_j i_{j+1}}$$

subject to

$$\begin{aligned}
\theta^k_{i_j i_{j+1}} &\geq \tau_{i_j i_{j+1}}(1 - \sigma^k_{i_j i_{j+1}}) && && (3.6)\\
\gamma^k_{i_j} + \theta^k_{i_j i_{j+1}} &\leq \gamma^k_{i_{j+1}} && \forall k \in \mathcal{K},\ j = 1, \ldots, |P_k| - 1, && (3.7)\\
e_k &\leq \gamma^k_{o_k} && \forall k \in \mathcal{K}, && (3.8)\\
\gamma^k_{|P_k|-1} + \theta_{i_{|P_k|-1} d_k} &\leq l_k && \forall k \in \mathcal{K}, && (3.9)\\
\gamma^{k_1}_i &= \gamma^{k_2}_i && \forall (k_1, k_2) \in J_{((i,t),(j,t'))}, \forall ((i,t),(j,t')) \in \overline{\mathcal{J}}, && (3.10)\\
\gamma^k_{i_j} &\geq 0 && \forall k \in \mathcal{K}, j = 1, \ldots, |P_k|, && (3.11)\\
\theta^k_{i_j i_{j+1}} &\geq \overline{\tau}^k_{i_j i_{j+1}}, && && \\
\sigma^k_{i_j i_{j+1}} &\in \{0, 1\} && \forall k \in \mathcal{K}, j = 1, \ldots, |P_k| - 1 && (3.12)
\end{aligned}$$

The objective is to minimize the number of arcs that are assigned a travel time shorter than the actual travel time. Constraints (3.6) count the number of arcs that are assigned a travel time shorter than the actual. Constraints (3.7) ensure the dispatch times are in accordance with the assigned travel times. Constraints (3.8) and (3.9) ensure that the dispatch times prescribed for a commodity enable it to depart from its origin after it becomes available and arrive at its destination before it is due. Constraints (3.10) ensure that all consolidations seen in the solution to SND($\mathcal{D}_\mathcal{T}$) are maintained.

We note that when the optimal value to this MIP is zero, the dispatch times γ^k_i show how to convert the solution to SND($\mathcal{D}_\mathcal{T}$) to a solution to CTSNDP of equal cost and SOLVE-CTSNDP can terminate. Conversely, when the optimal value is greater than zero, we choose to lengthen arcs (i_j, i_{j+1}) such that $\sigma^k_{i_j i_{j+1}} = 1$ for some k. We also note that we can speed up the MIP (without invalidating it) by fixing to 0 variables $\sigma^k_{i_j i_{j+1}}$ associated with arcs in the solution to SND($\mathcal{D}_\mathcal{T}$) that already have the correct length (i.e., are not "short").

3.4.4 Deriving a solution to CTSNDP from a solution to SND($\mathcal{D}_\mathcal{T}$)

When a solution to SND($\mathcal{D}_\mathcal{T}$) cannot be converted to a solution to CTSNDP of equal cost, we can still use it to construct a feasible solution of greater cost. Like the integer program

above, we seek to find dispatch times for every commodity k at every node in P_k such that (1) the commodity's availability time and due time are respected, and (2) commodities that are dispatched together in the optimal solution to SND($\mathcal{D}_\mathcal{T}$) are still dispatched together as much as possible (so that the same consolidations are realized). However, unlike the integer program above, we now ensure that the dispatch times respect the actual travel times of the arcs.

To determine these dispatch times we solve a linear program that is similar to the integer program above. We again use the variables $\gamma_{i_j}^k$, but now, for each pair of commodities $(k_1, k_2) \in J_{((i,t),(j,t'))}$, we define a variable $\delta_{ijt}^{k_1 k_2} \geq 0$ to capture any difference in dispatch time of the two commodities on arc (i, j). With these variables we then solve the following linear program (LP):

$$Z = \text{minimize} \sum_{((i,t),(j,t')) \in \overline{\mathcal{J}}} \sum_{(k_1,k_2) \in J_{((i,t),(j,t'))}} \delta_{ijt}^{k_1 k_2}$$

subject to

$$\begin{aligned}
\gamma_{i_j}^k + \tau_{i_j i_{j+1}} &\leq \gamma_{i_{j+1}}^k && \forall k \in \mathcal{K},\ j = 1, \ldots, |P_k| - 1, && (3.13)\\
e_k &\leq \gamma_{o_k}^k && \forall k \in \mathcal{K}, && (3.14)\\
\gamma_{i_{|P_k|-1}} + \tau_{i_{|P_k|-1} d_k} &\leq l_k && \forall k \in \mathcal{K}, && (3.15)\\
\delta_{ijt}^{k_1 k_2} &\geq \gamma_i^{k_1} - \gamma_i^{k_2} && \forall (k_1, k_2) \in J_{((i,t),(j,t'))}, \forall ((i,t),(j,t')) \in \overline{\mathcal{J}}, && (3.16)\\
\delta_{ijt}^{k_1 k_2} &\geq \gamma_i^{k_2} - \gamma_i^{k_1} && \forall (k_1, k_2) \in J_{((i,t),(j,t'))}, \forall ((i,t),(j,t')) \in \overline{\mathcal{J}}, && (3.17)\\
\gamma_{i_j}^k &\geq 0 && \forall k \in \mathcal{K}, j = 1, \ldots, |P_k|. && (3.18)
\end{aligned}$$

Because the optimal solution to SND($\mathcal{D}_\mathcal{T}$) satisfies constraints (3.5), there will always be a feasible solution to the LP. The only reason the consolidations seen in the solution to SND($\mathcal{D}_\mathcal{T}$) cannot be seen in a feasible solution to the CTSNDP is if a commodity participating in a consolidation travels on a path that contains an arc that is too short.

Note that neither the MIP or LP presented above require that the consolidations take place at the times "suggested" by the solution to SND($\mathcal{D}_\mathcal{T}$). It only stipulates that the commodities have to follow the same path and that the consolidations that occurred are

reproduced (as much as possible). However, the dispatch times γ_i^k prescribed by the solution to the LP represent a feasible solution to CTSNDP. Let the value of this solution be $z(P\text{-}CTSNDP)$. Thus, at each iteration of the algorithm, we can calculate an optimality gap with the following formula

$$\frac{z(P\text{-}CTSNDP) - z(\mathcal{D}_{\mathcal{T}})}{z(P\text{-}CTSNDP)}. \tag{3.19}$$

This also allows us to specify an optimality tolerance as a stopping condition when executing SOLVE-CTSNDP.

3.4.5 Comparison with the time bucket formulation for TSPTW

We finish this section by more closely considering the branch-and-cut algorithm of Dash et al. (2012) for the Traveling Salesman Problem with Time Windows (TSPTW), because it also employs dynamic discretization discovery ideas. Dash et al. (2012) present a formulation of the TSPTW that is based on partitioning the time windows into subwindows or *buckets*. The strength of the LP relaxation of the time bucket formulation depends on the partition of the time windows. In general, having more buckets results in stronger LP relaxations, but also in larger LP relaxations. To develop an efficient branch-and-cut algorithm, it is therefore important to strike the right balance between the strength of the LP relaxation and the time it takes to solve the LP relaxation. To obtain a "good" partition of the time windows, Dash et al. (2012) employ an iterative linear programming-based partition refinement scheme, i.e., the scheme dynamically discovers an appropriate partition (or discretization) of the time windows. To enhance the iterative refinement scheme, a *bucket graph* is constructed in which arcs $((i, b_i), (j, b_j))$ indicate that it is possible to reach bucket b_j at node j from bucket b_i at node i, and bucket preprocessing techniques are employed. This bucket graph can equivalently be viewed as a partially time-expanded network satisfying PROPERTIES 1 – 4. (A fully time-expanded network would correspond to partition of the time windows in subwindows of length one.) Thus, Dash et al. (2012) also manipulate a partially time-expanded network.

However, there are critical differences between their work and ours, in terms of the problems studied and the solution approaches developed. Regarding problems studied, we note that (1) in the TSPTW it is possible to restrict the searching for solutions to those without unforced waiting time, whereas in the CTSNDP waiting is often critical to achieve consolidations, and (2) in the CTSNDP it is trivial to find feasible solutions, whereas finding feasible solutions for the TSPTW is NP-hard (Savelsbergh, [1986]). Regarding solution approaches, the major differences are that (1) Dash et al. (2012) employ dynamic discretization discovery as a preprocessing scheme (once a partition has been established, it is never changed during the branch-and-cut search), whereas we continue to refine the discretization until the solution to SND($\mathcal{D}_{\mathcal{T}}$) can be converted to an optimal solution to CTSNDP, (2) Dash et al. (2012) use information from the solution to an LP to heuristically refine the set of time points, whereas we use information from the solution to an IP to carefully refine the set of time points to guarantee convergence to an optimal solution to CTSNDP, and (3) we focus much more strongly on keeping the number of time points in the partially time-expanded network to a minimum.

3.5 A Computational Study

The goal of the computational study presented in this section is to demonstrate the effectiveness and efficiency of the (straightforward implementation of the) proposed iterative refinement algorithm for solving CTSNDP and to gain a better understanding of the factors that contribute to its performance. We first describe the instances used in the computational study (Section 3.5.1), then we illustrate some of the challenges associated with discretizing time (Section 3.5.2), and then, we present the results of a series of experiments that demonstrate the efficacy of the proposed algorithm (Section 3.5.3).

To be able to assess the performance of the proposed algorithm on an instance, we also solve the instance using the formulation with full time discretization. We will refer to this as using *Full-Discretization*, or, FD. (We note that the same preprocessing techniques are

used when using Full-Discretization as when solving SND($\mathcal{D}_{\mathcal{T}}$)). Abusing terminology, we will sometimes use FD to refer to the integer program it solves.

3.5.1 Instances

We derive the instances used in our computational study from the C and C+ instances described in detail in Crainic, Frangioni, and Gendron, (2001). These instances have been used to benchmark the performance of many algorithms for the capacitated fixed charge network design problem (Ghamlouch, Crainic, and Gendreau, [2003]; Crainic, Gendron, and Hernu, [2004]; Ghamlouch, Crainic, and Gendreau, [2004]; Katayama, Chen, and Kubo, [2009]; Hewitt, Nemhauser, and Savelsbergh, [2010]; Yaghini, Rahbar, and Karimi, [2012]; Hewitt, Nemhauser, and Savelsbergh, [2013]). The instances vary with respect to the number of nodes (20, 30), arcs (230,300,520,700), commodities (40,100,200, and 400), whether the variable costs, c_{ij}, outweigh the fixed costs, f_{ij}, and whether the arcs are loosely or tightly capacitated. The results we present next are based on the 24 instances with 100, 200, or 400 commodities. The other, smaller instances are solved nearly instantaneously, and, thus, do not yield insights into the performance of our algorithm. We provide a detailed list of these instances in Table 3.1. We refer to these instances as "untimed" as they do not have any time attributes, e.g., there are no travel times associated with arcs or available and due times associated with commodities.

We "timed" these instances using the following scheme. First, for a given parameter ν, we set the travel time in minutes, τ_{ij}, of arc (i, j) to be proportional to its fixed charge. Specifically, we set $\tau_{ij} = \nu f_{ij} \ \ \forall (i, j) \in \mathcal{A}$. We calculated the value ν based on the premise that f_{ij} represents the transportation cost for a carrier that spends \$.55 cents per mile and their trucks travel at 60 miles per hour.

When commodities become available and when they are due partially dictates whether they can consolidate. To be able to measure the degree to which these parameters impact consolidation, we generate for each untimed instance with associated arc travel times

multiple instances with varying values for commodity available and due times. More specifically, we first calculate for each commodity $k \in \mathcal{K}$ the length of the shortest path from o_k to d_k with respect to the travel times τ_{ij}. We call this length $\mathcal{L}_k$ and the average of these $|\mathcal{K}|$ lengths $\mathcal{L}$. We then create three normal distributions from which we draw the available time for each commodity, all of which are defined by a mean, μ_e, of $\mathcal{L}$ minutes, but vary with respect to their standard deviation. Specifically, we consider three values for the standard deviation, σ_e: $\frac{1}{3}\mathcal{L}$, $\frac{1}{6}\mathcal{L}$, and $\frac{1}{9}\mathcal{L}$. Given commodity k's available time, e_k, at its origin, it can arrive at its destination no sooner than $e_k + \mathcal{L}_k$. Next, we introduce for each commodity $k \in \mathcal{K}$ a time flexibility, $f_k \geq 0$, and set its due time, l_k, to $e_k + \mathcal{L}_k + f_k$. Similar to determining the available times, we create two normal distributions from which we draw these time flexibilities. The first has a mean, μ_f, of $\frac{1}{2}\mathcal{L}$, and the second has $\mu_f = \frac{1}{4}\mathcal{L}$. Both distributions have a standard deviation, σ_f of $\frac{1}{6}\mu_f$.

In summary, there are three normal distributions from which we draw commodity available times and two normal distributions from which we draw commodity time flexibility. As such, we have six sets of instances, one for each combination of distributions, and randomly generate three instances for each set. When we randomly sample from one of the distributions, we repeatedly draw from the distribution until we generate a value that falls within three standard deviations of the mean of the distribution.

Therefore, we have a total of $24 \times 6 \times 3 = 432$ instances. Finally, we consider five discretization parameters, Δ: 60 minutes, 30 minutes, 15 minutes, 5 minutes, and 1 minute. We summarize the parameter values used to generate the instances used in our computational study in Table 3.2.

Table 3.1: "Flat" instances from Crainic, Frangioni, and Gendron, (2001) used in study

Instance	$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	$\|\mathcal{K}\|$	Fixed or Variable cost	Tight or Loosely capacitated
c37	20	230	200	V	L
c38	20	230	200	F	L
c39	20	230	200	V	T
c40	20	230	200	F	T
c45	20	300	200	V	L
c46	20	300	200	F	L
c47	20	300	200	V	T
c48	20	300	200	F	T
c49	30	520	100	V	L
c50	30	520	100	F	L
c51	30	520	100	V	T
c52	30	520	100	F	T
c53	30	520	400	V	L
c54	30	520	400	F	L
c55	30	520	400	V	T
c56	30	520	400	F	T
c57	30	700	100	V	L
c58	30	700	100	F	L
c59	30	700	100	V	T
c60	30	700	100	F	T
c61	30	700	400	V	L
c62	30	700	400	F	L
c63	30	700	400	V	T
c64	30	700	400	F	T

Table 3.2: Time-oriented characteristics

Normal distribution	μ	σ
For generating e_k	$\mathcal{L}$	$\frac{1}{3}\mathcal{L}$, $\frac{1}{6}\mathcal{L}$, and $\frac{1}{9}\mathcal{L}$
For generating f_k	$\frac{1}{2}\mathcal{L}$, $\frac{1}{4}\mathcal{L}$	$\frac{1}{6}\mu$
Δ	60 min, 30 min, 15 min, 5 min, 1 min	

3.5.2 The impact of discretizing time

As mentioned in the introduction, the granularity of the time discretization has an impact on the accuracy of a formulation as well as its size.

With regard to the size of the formulation, we report in Figure 3.4 the growth in the full discretization integer program, in terms of the number of variables and the number of constraints, as the granularity is refined, i.e., when Δ is changed from 60 to 30, from 60 to 15, from 60 to 5, and from 60 to 1. (We report averages over all instances.) We see that the growth is substantial. Refining the granularity from a 60-minute discretization to a 1-minute discretization results in a factor 15 increase in the size of the integer program.

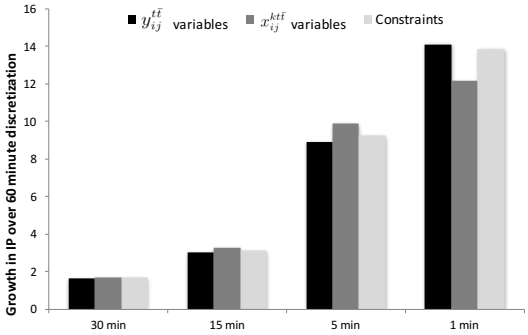


Figure 3.4: Growth in FD when the discretization Δ is changed from 60 to a smaller value.

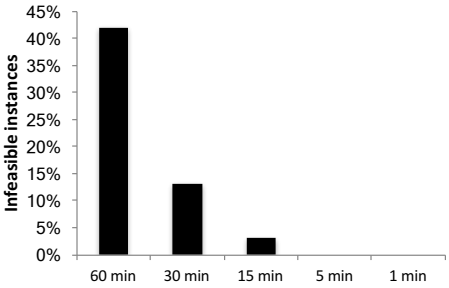


Figure 3.5: The % of instances that become infeasible due to discretization for different choices of Δ.

With regard to the accuracy, consider the time-feasibility of a path p in the flat network. A path p in the flat network is time-feasible if $\sum_{(i,j)\in p} \tau_{ij} \leq l_k - e_k$. However, for discretization Δ, the travel time of an arc (i,j) will be modeled as $\Delta\lceil \tau_{ij}/\Delta \rceil$, which can be strictly greater than τ_{ij}, the available time of a commodity k will be modeled as $\Delta\lceil e_k/\Delta \rceil$, which can be strictly greater than e_k, and the due time of a commodity k will be modeled as $\Delta\lfloor l_k/\Delta \rfloor$, which can be strictly smaller than l_k. As a result, paths that are time-feasible when considering the true travel times and the true available and due time may be rendered infeasible for a discretization Δ. Furthermore, if all time-feasible paths for some commodity k in an instance are rendered infeasible for a discretization Δ, then the

instance itself will become infeasible. That this is a relevant and important issue is shown in Figure 3.5, where we display the percentage of the 432 instances that become infeasible due to discretization for different choices of Δ. We see that for $\Delta = 60$ over 40% of the instances become infeasible (when they are in fact feasible for a 1-minute discretization).

Figures 3.4 and 3.5 highlight the fundamental issue with modeling time-indexed decisions; while finer discretizations of time lead to more accurate models, an enumerative approach to choosing time points to model can lead to significantly larger optimization problems.

3.5.3 Performance of SOLVE-CTSNDP

We conducted a set of a computational experiments to assess the performance of our implementation of the proposed dynamic discretization discovery algorithm for solving CTSNDP using the instances that were not rendered infeasible by discretization (recall Figure 3.5 in the previous subsection). In all experiments, we gave FD and SOLVE-CTSNDP the same stopping criteria: a proven optimality gap of less than or equal to 1%, where the optimality gap is calculated using (3.19), or a maximum run-time of two hours. Note that when SOLVE-CTSNDP stops because the feasible solution to the relaxation, $SND(\mathcal{D}_{\mathcal{T}})$, can be converted to a feasible solution to CTSNDP, it terminates with a provably optimal solution. All experiments were run on a cluster of computers and each job was allowed to use a maximum of 16GB of memory. Experiments were run on a cluster of nodes containing 32 cores each, with speeds ranging from 2.3 to 2.8GHZ. Each node has 256G of RAM.

To compare the performance of FD and SOLVE-CTSNDP, we graph averages over instances with the same discretization parameter Δ of: the time to termination (Figure 3.6), the optimality gap at termination (Figure 3.7), and the percentage of instances solved (Figure 3.8). We note that SOLVE-CTSNDP never exceeds the 16 GB memory limit, whereas FD does so for 167 of the 432 (nearly 39%) of the instances with $\Delta = 1$ (and

never for the other discretizations). Therefore, we do not display results for $\Delta = 1$ in Figures 3.6 and 3.7. Instead, for the instances with $\Delta = 1$, we report in Table 3.3 the performance of both methods separately for the instances where FD does not exceed the 16 GB memory limit (FD $\leq$ 16 GB) and the instances where FD does exceed the 16 GB memory limit (F-D $>$ 16 GB).

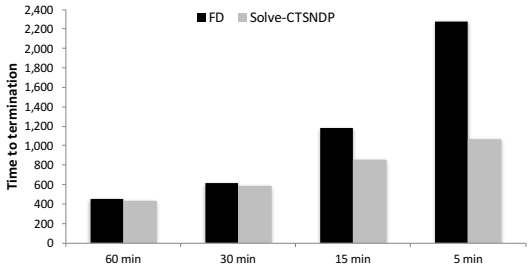


Figure 3.6: Time to termination for different Δ

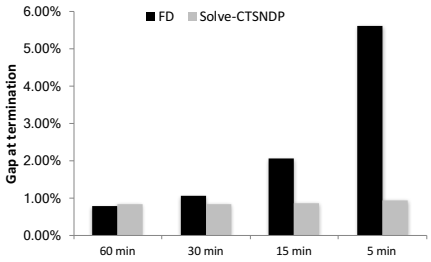


Figure 3.7: Optimality gap at termination for different Δ

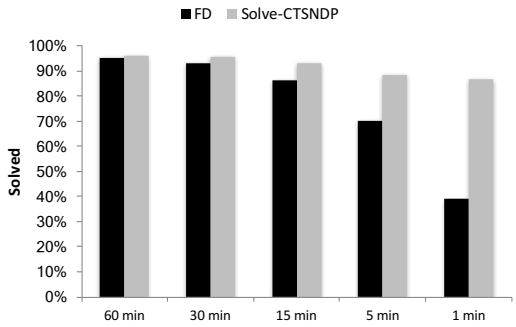


Figure 3.8: Fraction of instances solved within the memory and time limits for different Δ

Table 3.3: Performance when $\Delta = 1$ minute

Instances	Method	Time	Opt. Gap	Solved
FD $\leq$ 16GB	FD	3,097.832	3.85%	62.26%
	SOLVE-CTSNDP	417.04	0.78%	98.87%
FD $>$ 16GB	SOLVE-CTSNDP	3,106.49	1.33%	67.66%

We see that the performance of SOLVE-CTSNDP is comparable to that of FD on instances with coarse discretizations (60 min and 30 min), but clearly outperforms FD on fine discretizations (15 min, 5 min, and 1 min), where it requires less time to solve more instances and when it cannot solve an instance yields a smaller optimality gap. Thus, as

expected, the performance of FD significantly degrades as the discretization becomes finer, but, as anticipated, SOLVE-CTSNDP remains effective. We complement the averages reported in the previous figures with distributions (in deciles) for each discretization of the the time to termination (Figure 3.9) and the optimality gap at termination (Figure 3.10) for SOLVE-CTSNDP. In these figures we see that the distribution for the time to termination is positively skewed, with outliers pulling the average up. For example, for all discretizations the 60^{th} percentile for the time to termination is less than seven minutes (and significantly less than the average). We note that for all discretizations the 80^{th} percentile of optimality gaps is within our optimality tolerance of 1%.

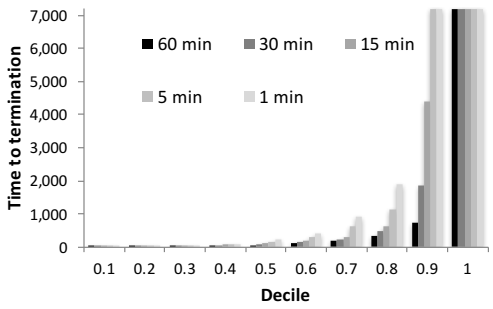


Figure 3.9: Time to termination.

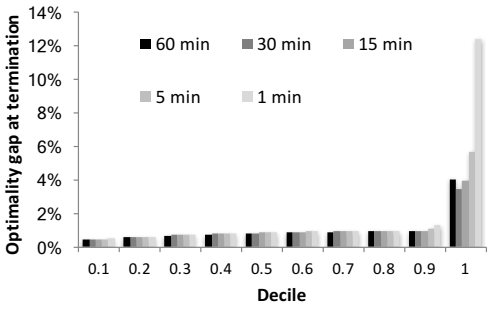


Figure 3.10: Optimality gap at termination.

To better understand why SOLVE-CTSNDP outperforms FD, we first compare $|\mathcal{N}_{FD}|$, the cardinality of the node set of the fully time-expanded network that forms the basis of the integer program solved by FD, and $|\mathcal{N}_{\mathcal{T}}|$, the node set of the partially time-expanded network that forms the basis of the *last* integer program solved by SOLVE-CTSNDP. In Figure 3.11, we show for the instances with a given discretization parameter Δ and for a given ratio r ($0 < r < 1$), the fraction of instances with $|\mathcal{N}_{\mathcal{T}}|/|\mathcal{N}_{FD}| \leqslant r$. We see that SOLVE-CTSNDP works with significantly smaller time-expanded networks while searching for a provably optimal solution to CTSNDP than FD, especially for instances with $\Delta = 1$, where $|\mathcal{N}_{\mathcal{T}}|/|\mathcal{N}_{FD}| \leqslant 0.04$ for *all* instances.

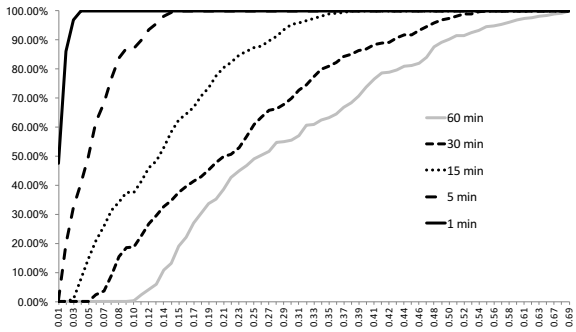


Figure 3.11: Relative time-expanded network size of the final SND($\mathcal{D}_{\mathcal{T}}$).

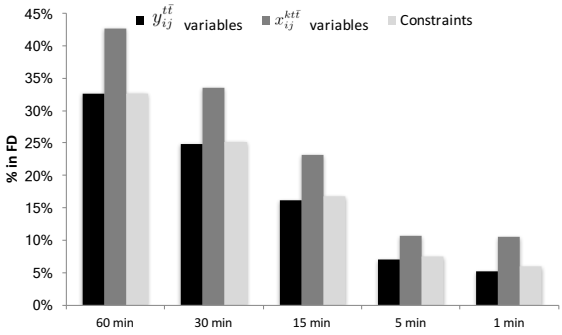


Figure 3.12: Relative integer programming size associated with the final SND($\mathcal{D}_{\mathcal{T}}$).

Next, in Figure 3.12, we compare the size of the integer programs solved by FD and the size of the last integer programs solved by SOLVE-CTSNDP (i.e., of the integer program associated with the final SND($\mathcal{D}_{\mathcal{T}}$)). Specifically, we show for the instances with a given discretization parameter Δ, the averages of the following ratios: the number of $y_{ij}^{t\bar{t}}$ variables in the last integer program solved by SOLVE-CTSNDP and the number of $y_{ij}^{t\bar{t}}$ variables in the integer program solved by FD, the number of $x_{ij}^{kt\bar{t}}$ variables in the last integer program solved by SOLVE-CTSNDP and the number of $x_{ij}^{kt\bar{t}}$ variables in the integer program solved by FD, and the number of constraints in the last integer program solved by SOLVE-CTSNDP and the number of constraints in the integer program solved by FD. We see that the last integer programs solved by SOLVE-CTSNDP are significantly smaller than those solved by FD. Furthermore, as expected, we see that the finer the discretization the smaller the relative size of the integer program associated with the final SND($\mathcal{D}_{\mathcal{T}}$) solved by SOLVE-CTSNDP.

In Figure 3.11, we saw that the cardinality of the node set of the partially time-expanded network of the last SND($\mathcal{D}_{\mathcal{T}}$)) solved by SOLVE-CTSNDP is much smaller than the cardinality of the node set of the fully time-expanded network for the same instance. Next, we explore the growth of the partially time-expanded networks during the execution of SOLVE-CTSNDP. More specifically, in Figure 3.13, we report for the instances with a given discretization parameter Δ the averages of $|\mathcal{N}_{\mathcal{T}}|/|\mathcal{N}_{FD}|$ by iteration of SOLVE-

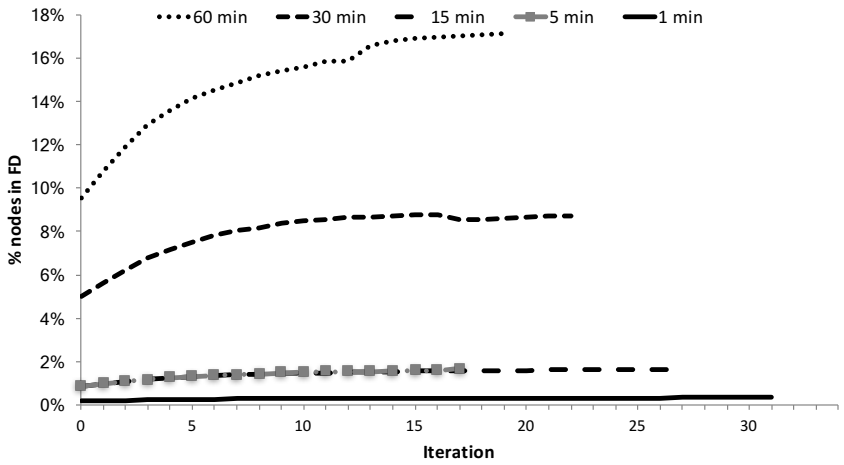


Figure 3.13: Relative time-expanded network size by iteration for instances with $|\mathcal{N}| = 20, |\mathcal{A}| = 200, |\mathcal{K}| = 200, \sigma_e = \frac{1}{9}\mathcal{L}, \mu_f = \frac{1}{2}\mathcal{L}$.

CTSNDP for a specific flat network and set of timing parameter values (this set of instances was chosen because SOLVE-CTSNDP struggled the most with them, solving the smallest fraction of instances).

We see that the size of the partially time-expanded networks created and refined by SOLVE-CTSNDP remains fairly stable during the execution of algorithm (even for instances with discretization $\Delta = 60$, the relative size only increases from about 10% to about 18%).

The results of the computational experiments discussed above clearly demonstrate SOLVE-CTSNDP's superiority over FD. We conclude that SOLVE-CTSNDP outperforms FD because it starts with a significantly smaller time-expanded network than FD and refines it in such a way that the time-expanded network grows only modestly. As a result, the integer programs solved by SOLVE-CTSNDP are significantly smaller than those solved by FD.

Next, we analyze the changes in the lower and upper bounds during the execution of SOLVE-CTSNDP. We note that because the number of iterations required to solve an instance varies across the instances, the number of instances for which we have a lower and upper bound at the k^{th} iteration is typically greater than the number of instances for

which we have a lower and upper bound at the $k + 1^{th}$ iteration and this has to be kept in mind when interpreting the average values reported for an iteration.

In Figure 3.14, we report averages over all instances, but by discretization parameter and iteration, of gaps for both these measures. Specifically, we report the gap in value between the primal solution produced by SOLVE-CTSNDP at an iteration and the primal solution produced at termination (labeled Δ min primal). Similarly, we report the gap in value between the dual bound produced by SOLVE-CTSNDP at an iteration and the dual bound produced at termination (labeled Δ min dual). (We note again that because not all executions of SOLVE-CTSNDP require the same number of iterations to terminate, for each iteration, we are reporting averages over the instances that needed at least that many iterations to terminate.) First, we observe that both gaps converge fairly quickly, with

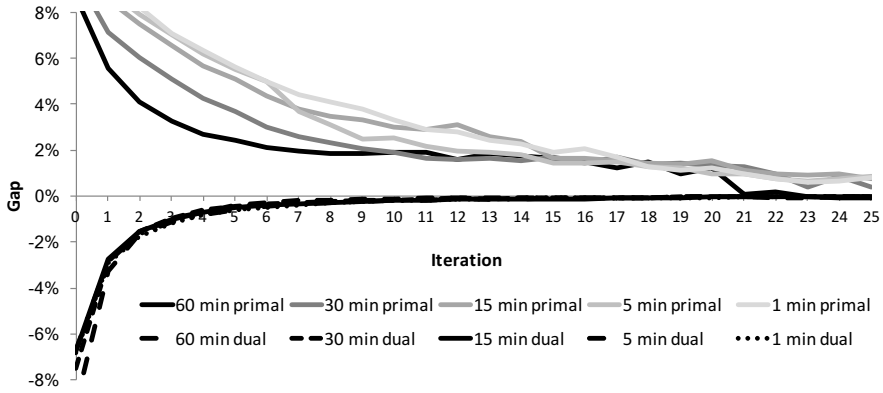


Figure 3.14: Primal and dual gaps by iteration.

the dual bound converging more quickly than the objective function value of the primal solution. Importantly, we note that the performance of SOLVE-CTSNDP with respect to both the quality of the primal solution and the strength of the dual bound produced at an iteration is consistent across discretizations. Coupled with the results discussed previously, we conclude that SOLVE-CTSNDP is robust with respect to the discretization parameter, Δ.

3.5.4 Potential of SOLVE-CTSNDP in practice

As a final test of the effectiveness of the algorithm, we created five instances using data from a large less-than-truckload freight transportation carrier in the United States. While this carrier operates in nearly all states, we limited our instances to activities in the Northwest region of the United States. We construct the instances using five days worth of freight. When optimizing how freight moves within this region, we include freight that originates outside the region but destined for a terminal inside the region and freight that originates inside the region but destined for a terminal outside the region. This is done by exploiting knowledge of the carrier's planned path for routing the freight. As an example, consider freight that originates on Day 1 in Miami, FL, and is destined for Seattle, WA on Day 5. The carrier's planned path for that freight enters the Northwest in Boise, ID on Day 3. We model that freight as freight that originates in Boise on Day 3 and is due in Seattle on Day 5.

Due to the operational practices of this particular carrier, these instances differ in several ways from those used in our earlier computational experiments. Specifically, it is assumed that on the day that freight becomes available, it becomes available at 7 pm at its origin terminal (as that is when the carrier expects trucks to return from pickup and delivery routes). Thus, by considering a five day planning horizon, freight only originates at five different time points. Similarly, it is assumed that on the day freight is due at the customer destination, it must be at the associated terminal at 8 am (as that is when the carrier needs it to be ready for the pickup and delivery route). Finally, we note that the carrier quotes service (how long freight will take from origin to destination) to customers in terms of days. Thus, the time the carrier has to deliver freight also has a discrete nature. Based on discussions with the carrier, it was determined that 30 minutes is the finest discretization that would yield plans that could be implemented in practice. We ran both SOLVE-CTSNDP and FD for two hours on each instance and report the results of their execution, as well as data regarding each instance, in Table 3.4. We calculate "Delivery

window (avg.)" as $\frac{\sum_{k\in\mathcal{K}} l_k - e_k}{|K|}$, while "Planning horizon" is calculated as $\max_k(l_k - e_k)$. Finally, "Freight capacity fraction (avg.)" is calculated as

$$\frac{\sum_{k\in\mathcal{K}} q_k}{|K|} \Big/ \frac{\sum_{(i,j)\in\mathcal{A}} u_{ij}}{|A|}.$$

We report gaps in terms of objective function value of the primal solutions produced by each method. We calculate "Primal gap" as $(z_{CTSNDP} - z_{FD})/z_{CTSNDP})$, where z_X represents the objective function value of the solution produced by method X.

Table 3.4: Performance of SOLVE-CTSNDP on instances derived from data from a US LTL carrier.

States	$\|N\|$	$\|A\|$	$\|K\|$	Delivery window (avg.)	Planning horizon	Freight capacity fraction (avg.)	SOLVE-CTSNDP Time to termination	SOLVE-CTSNDP Optimality gap	FD Time to termination	FD Optimality gap	Primal gap
ID,MT,OR,WA	10	54	224	34.22	240	1.25	30	0.92%	213	0.94%	0.26%
CO,ID,MT,OR,WA	14	81	341	49.07	240	1.05	7,200	1.68%	7,200	3.63%	-1.29%
CO,ID,MT,OR,WA,NV	16	109	469	52.11	240	0.94	7,200	2.74%	7,200	25.28%	-28.26%
CO,ID,MT,OR,WA,UT	15	104	458	50.28	240	1.00	7,200	1.45%	7,200	19.71%	-20.64%
ID,MT,OR,WA, NV,UT	13	97	429	43.88	240	1.05	7,200	3.00%	7,200	23.78%	-25.18%

We see that SOLVE-CTSNDP easily solves the instance with only four states, but is unable to solve the larger instances (to optimality). However, it is able to produce a primal solution and a dual bound with a relatively small (provable) optimality gap. We also note that for the larger instances, SOLVE-CTSNDP reports an optimality gap of between 3 and 4 percent after a little over an hour, suggesting the algorithm may be used as a heuristic. Similarly, FD is only able to solve the smallest instance. However, FD produces a much larger provable optimality gap than SOLVE-CTSNDP after two hours on the larger instances, and, looking at the "Primal gap", we see that SOLVE-CTSNDP is able to produce higher-quality solutions than FD.

Regarding our previous experiments, we note that for the instances generated with $\Delta = 30$, the average "Delivery Window" is 54.65, which is slightly larger than the average for the real-world-inspired instances. However, the planning horizon of the real-world-inspired instances, i.e., 240, is much longer than the planning horizon of the instances used in the previous experiments, i.e., 122 periods. Finally, we note that the commodities in the instances used in the previous experiments are, relatively-speaking, much smaller, as the average "Freight capacity fraction" for those instances is 0.16. We attribute the difficulty

that SOLVE-CTSNDP and FD have in solving the two larger real-world-inspired instances to the large number of commodities and the length of the planning horizon as both have a direct impact on the size of the integer program solved at each iteration.

3.6 Conclusions and Future Work

We have presented an algorithm for solving service network design problems that uses time-expanded networks, but that avoids having to introduce approximations (and thus uncertainty about the quality of the solution) to keep the size of the time-expanded network manageable.

Because time-expanded networks are commonly used in the solution of many transportation problems, we hope and expect that many of the fundamental ideas underlying our approach can be applied in other contexts as well.

Our computational study has demonstrated that the current implementation of SOLVE-CTSNDP is quite effective. However, there are many enhancements that will increase its efficiency. We mention only a few. First, it is unnecessary to solve SND($\mathcal{D}_{\mathcal{T}}$), which is the most time-consuming step in the algorithm, to optimality at each iteration. In the initial iterations, it may suffice to only solve the linear programming relaxation or to solve the problem to withing a proven percentage of optimality. Second, if SND($\mathcal{D}_{\mathcal{T}}$) does not change much from one iteration to the next, it may be possible to construct a high-quality initial solution to speed up the solution time.

The ability to solve very large instance of a service network design problem also opens up the possibility to study new and innovative time-focused service network design models. For example, a carrier may be interested in understanding the degree to which altering the available and/or due times of a commodity impacts the costs. Such decisions can easily be incorporated in a service network design model, but will increase the size of instance substantially. However, by using an iterative refine algorithm based on partially time-expanded networks, the increase in size may be overcome.

Finally, we note that many real-world service network design problems are computationally intractable due to their scale on two dimensions: (1) time, and, (2) number of shipments. For the carrier that inspired this work, modeling a week's worth of actions in their network on a time-space network based on a 30 minute discretization of time would yield a network with nearly 40,000 nodes and over 1.2 million arcs. We believe that the algorithm presented in this paper presents a significant step forward in terms of tackling this explosion in size. However, the number of shipments (which would map to commodities in a service network design model) for the carrier over a five day span would exceed 60,000. Such a large number of commodities would likely render the integer programs solved by the algorithm presented in this paper computationally intractable. As a result, in future work, we intend to develop a similar algorithm for an explosion in size along this dimension.

CHAPTER 4
INTERVAL-BASED DYNAMIC DISCRETIZATION DISCOVERY

4.1 Introduction

In this chapter, we introduce an alternate algorithm for solving the Continuous-Time Service Network Design Problem (CTSNDP). The high-level concept is similar to what is presented in Chapter 3, that is, both algorithms iteratively refine a time-expanded network, however, this alternate approach has some fundamental differences. As the name suggests, the interval-based dynamic discretization discovery method (DDDI) considers for each location, a partition of the time horizon into non-uniform intervals of time. Nodes in the corresponding time-expanded network are defined by *node-intervals*, that is (location, interval) pairs. We define strict conditions for connecting two node-intervals, known as a *timed-arc*, such that the associated integer program is guaranteed to be a lower bound, and, with a suitably chosen refinement process, yields a continuous-time optimal solution. The process of refining the discretization is considered then as the partitioning of an interval, whereas for contrast, DDD refines the discretization by lengthening arcs. Note that the differences between DDD and DDDI are more than just conceptual, for instance, none of the four main properties required by DDD (Property 1-4 in Chapter 3) are supported by DDDI. As such, we will present the DDDI algorithm independently from DDD, with its own notation. For a full analytic and empirical comparison between the two approaches, see Section 4.8.

One of the driving motivations of the interval-based approach is to keep the corresponding time-expanded network as small as possible. This principle is also extended to the refinement step, which attempts to exploit structures in the solution in order to converge efficiently, while keeping growth small. In this way, due to the smaller associated integer

programs, we hope that it is possible to solve larger instances, with faster solution times.

4.2 Problem Description

Recall from Section 2.2, that in the CTSNDP the goal is to transport a set of commodities K (delivering each before its due time), across network $G = (N, A)$ so as to minimize the total cost of transport. Each commodity $k \in K$ has origin o^k, destination d^k, arrival/early time e^k, due/late time l^k, and quantity q^k. The network has locations, N, and transportation options, A. Each arc $a \in A$ has an associated transit time τ_a, capacity u_a, fixed cost f_a, and variable cost v_a. Note that all input is assumed to be rational, and so it suffices to consider the integer case.

Let H be the time horizon for the instance, where $H \geq \max_{k \in K} \{l^k\}$. We define a *discretization* as a finite set of (node, time) pairs, i.e. $\mathcal{T} \subset N \times [0, H]$. Each commodity $k \in K$ has an associated time-expanded network, $\mathcal{G}^k_{\mathcal{T}} = \left(\mathcal{N}_{\mathcal{T}}, \mathcal{A}^k_{\mathcal{T}}\right)$, generated by the time points $\mathcal{T}$. Node-intervals are defined as $(n, t, t') \in \mathcal{N}_{\mathcal{T}} \subseteq N \times [0, H]^2$, and timed-arcs as $(i_1, i_2) = ((n_1, t_1, t'_1), (n_2, t_2, t'_2)) \in \mathcal{A}^k_{\mathcal{T}} \subseteq \mathcal{N}_{\mathcal{T}} \times \mathcal{N}_{\mathcal{T}}$. The *system* of time-expanded commodity networks for an instance, discretized by $\mathcal{T}$, is defined as $\mathcal{G}^K_{\mathcal{T}} = \{(\mathcal{N}_{\mathcal{T}}, \mathcal{A}^k_{\mathcal{T}}) : k \in K\}$.

Let $\mathcal{A}^K_{\mathcal{T}} = \bigcup_{k \in K} \mathcal{A}^k_{\mathcal{T}}$ be the set of timed-arcs which exist for at least one commodity in $\mathcal{G}^K_{\mathcal{T}}$. Note that $\mathcal{A}^K_{\mathcal{T}}$ contains both holding arcs (that link consecutive time-intervals at the same location), and dispatch arcs (linking time-intervals between different locations). Then, for ease of notation, let $\tau_a := \tau_{(n_1,n_2)}$, and similarly for f_a, v_a, u_a, for $a = ((n_1, t_1, t'_1), (n_2, t_2, t'_2)) \in \mathcal{A}^K_{\mathcal{T}}$. Also, let $\mathcal{D}^k_{\mathcal{T}} \subseteq \mathcal{A}^k_{\mathcal{T}}$, and similarly $\mathcal{D}^K_{\mathcal{T}} \subseteq \mathcal{A}^K_{\mathcal{T}}$, be the set of dispatch arcs, that is, where $n_1 \neq n_2$.

We propose an iterative approach for refining a non-uniform time-discretization, such that the corresponding integer program yields an optimal solution to the continuous-time problem. The following model, CTSNDP($\mathcal{T}$), approximates CTSNDP for the given $\mathcal{G}^K_{\mathcal{T}}$ system. Note that this is parameterized by the discretization $\mathcal{T}$. Observe that CTSNDP can

be defined as CTSNDP($\mathcal{T}^*$), having the discretization $\mathcal{T}^*$ that is sufficiently fine to ensure that any solution of CTSNDP($\mathcal{T}^*$) is continuous-time optimal. CTSNDP($\mathcal{T}$) is defined as:

Sets

$\delta_i^{k-} \subseteq \mathcal{A}_{\mathcal{T}}^k$ arcs that flow into node-interval $i \in \mathcal{N}_{\mathcal{T}}$ for commodity $k \in K$

$\delta_i^{k+} \subseteq \mathcal{A}_{\mathcal{T}}^k$ arcs that flow out of node-interval $i \in \mathcal{N}_{\mathcal{T}}$ for commodity $k \in K$

Parameters

r_i^k supply / demand on the network with $r^k_{(n,t,t')} = \begin{cases} 1 & \text{if } n = o^k \text{ and } t \leq e^k < t' \\ -1 & \text{if } n = d^k \text{ and } t \leq l^k < t' \\ 0 & \text{otherwise} \end{cases}$

Decision Variables

x_a^k $\begin{cases} 1 & \text{if commodity } k \in K \text{ dispatches on arc } a \in \mathcal{A}_{\mathcal{T}}^k \\ 0 & \text{otherwise} \end{cases}$

z_a number of vehicles dispatched on arc $a \in \mathcal{A}_{\mathcal{T}}^K$

Model

$$\min \sum_{k \in K} \sum_{a \in \mathcal{A}_{\mathcal{T}}^k} v_a q^k x_a^k + \sum_{a \in \mathcal{D}_{\mathcal{T}}^K} f_a z_a, \tag{4.1a}$$

subject to

$$\sum_{a \in \delta_i^{k+}} x_a^k - \sum_{a \in \delta_i^{k-}} x_a^k = r_i^k, \qquad \forall k \in K, i \in \mathcal{N}_{\mathcal{T}}, \tag{4.1b}$$

$$\sum_{k \in K \,:\, a \in \mathcal{A}_{\mathcal{T}}^k} q^k x_a^k \leq z_a u_a, \qquad \forall a \in \mathcal{D}_{\mathcal{T}}^K, \tag{4.1c}$$

$$x_a^k \in \{0,1\}, \qquad \forall k \in K, a \in \mathcal{A}_{\mathcal{T}}^k, \tag{4.1d}$$

$$z_a \in \mathbb{Z}_+, \qquad \forall a \in \mathcal{D}_{\mathcal{T}}^K. \tag{4.1e}$$

The objective $(4.1a)$ minimizes the total cost (with both variable and fixed dispatch costs). The flow constraints $(4.1b)$ ensure that each commodity leaves its origin node after e^k and reaches its destination node before l^k. In order to dispatch all commodities traveling on the same timed-arc, the consolidation constraints $(4.1c)$ require that the number

of vehicles is sufficient, taking into account capacity u_a. Lastly, $(4.1d)$ and $(4.1e)$ define the variables and domains.

4.3 Constructing the Time-Expanded System

Time-expanded networks are typically built from timed-copies (of an untimed network), each corresponding to elements in a given discretization. To avoid confusion when talking about the timed/untimed networks and associated paths etc, we introduce the following terminology.

The **flat** network is defined as the original network $G = (N, A)$, as opposed to the time-expanded network $\mathcal{G}_{\mathcal{T}}^k = \left(\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}}^k\right)$, for some commodity k. A **flat-path** is a simple path on the flat network and can be viewed as either an ordered set of nodes or an ordered set of arcs. The path $P \subseteq A$ is k**-feasible** for commodity $k \in K$ if it is a flat-path from o^k to d^k that can be traversed so as to depart o^k no earlier than e^k and arrive at d^k no later than l^k, i.e., for which $e^k + \sum_{a \in P} \tau_a \leq l^k$. A **timed-path** is a simple path in the time-expanded network $\mathcal{G}_{\mathcal{T}}^k$, for some commodity $k \in K$, that, in addition, never visits a node $n \in N$ more than once (i.e. a visit is associated with a dispatch arc, travel along holding arcs does not imply new visits). A **shortest path** for a commodity is a flat-path from its origin node to its destination node that has the least total transit time. Furthermore, let $\gamma\left(n_1, n_2\right)$ be time of the shortest path (least transit time) from n_1 to n_2, with $n_1, n_2 \in N$. Note that shortest paths can be tightened for commodity k by enforcing paths to never arrive into o^k nor dispatch from d^k.

With this in mind, the system of time-expanded networks $\mathcal{G}_{\mathcal{T}}^K$, will be constructed from the following input:

- a flat network, $G = (N, A)$,
- a set of commodities, K, together with their associated origin and destination nodes, and their earliest dispatch and latest arrival times,

- a time horizon, H, and
- a finite set of node and time point pairs $\mathcal{T}$.

It is assumed that G is a directed graph with no self loops or multiple edges. We require that H is at least the latest arrival time of any commodity, and that $\{0, H\} \subseteq \mathcal{T}^n \subset [0, H]$ for every node $n \in N$.

4.3.1 Node-intervals

Let $\mathcal{T}^n$ be the set of time points for node $n \in N$, and $\mathcal{T}^n_{(i)}$ the ith time point of $\mathcal{T}^n$ in ascending order. Then the set of node-intervals is constructed by:

$$\mathcal{N}_{\mathcal{T}} = \left\{ \left(n, \mathcal{T}^n_{(i)}, \mathcal{T}^n_{(i+1)}\right) \,:\, n \in N,\, i = 1, \ldots, |\mathcal{T}^n| - 1 \right\}, \quad \text{where } \mathcal{T}^n_{(1)} = 0,\, \mathcal{T}^n_{(|\mathcal{T}^n|)} = H.$$

Proposition 2. *For any $\alpha \in \mathcal{T}^n$ and $(n, t, t') \in \mathcal{N}_{\mathcal{T}}$, if $\alpha < t'$ then $\alpha \leq t$*

Proof. Let $i, j \in \mathbb{Z}$ be such that $\alpha = \mathcal{T}^n_{(i)}$ and $t' = \mathcal{T}^n_{(j)}$, then $t = \mathcal{T}^n_{(j-1)}$ by construction of $\mathcal{N}_{\mathcal{T}}$. Thus $\alpha < t'$ implies $i < j$, and thus $i \leq j - 1$. Hence $\alpha \leq t$. □

4.3.2 Timed-arcs

The conditions that determine whether timed-arc $((n_1, t_1, t'_1), (n_2, t_2, t'_2))$ exists in $\mathcal{A}^k_{\mathcal{T}}$, for commodity $k \in K$, can be summarized in the box below. This notation extends A to allow self loops, with $(n, n) \in A$ and $\tau_{(n,n)} := 0$, and we say that a path uses (n, n) if it includes n. Details regarding the choice of these conditions will follow.

⟦1⟧ $(n_1, n_2) \in A$, and, if $n_1 = n_2$, then $t'_1 = t_2$,

⟦2⟧ there exists a k-feasible path that uses (n_1, n_2),

⟦3⟧ $e^k + \gamma\left(o^k, n_1\right) < \min\left\{t'_1,\ t'_2 - \tau_{(n_1,n_2)}\right\}$,

⟦4⟧ $\max\left\{e^k + \gamma\left(o^k, n_1\right) + \tau_{(n_1,n_2)},\ t_1 + \tau_{(n_1,n_2)},\ t_2\right\} + \gamma\left(n_2, d^k\right) \leq l^k$, and

⟦5⟧ $\tau_{(n_1,n_2)} < t'_2 - t_1$ and, if $n_1 \neq n_2$, then $t_2 - t'_1 < \tau_{(n_1,n_2)}$.

Note that, since a k-feasible path is simple, condition $[\![2]\!]$ ensures that if $n_1 \neq n_2$, then $n_1 \neq d^k$ and $n_2 \neq o^k$. To see why these conditions have been chosen, consider that the set of timed-arcs $\mathcal{A}^k_{\mathcal{T}}$ for commodity $k \in K$ is defined as

$$\mathcal{A}^k_{\mathcal{T}} = \mathcal{D}^k_{\mathcal{T}} \cup \mathcal{H}^k_{\mathcal{T}}$$

where $\mathcal{D}^k_{\mathcal{T}}$ is the set of dispatch-arcs for commodity k, and $\mathcal{H}^k_{\mathcal{T}}$ is the set of holding-arcs for commodity k, defined in the following sections.

4.3.2.1 Dispatch-arcs

The dispatch-arc $a = (i_1, i_2) \in \mathcal{N}_{\mathcal{T}} \times \mathcal{N}_{\mathcal{T}}$, connecting two node-intervals $i_1 = (n_1, t_1, t'_1)$ and $i_2 = (n_2, t_2, t'_2)$, for commodity k, is included in $\mathcal{D}^k_{\mathcal{T}}$, if there exists a k-feasible path that uses arc (n_1, n_2), and that can be traversed so as to depart from n_1 at some time in the interval $[t_1, t'_1)$, and arrive at n_2 in the interval $[t_2, t'_2)$, while still departing from o^k no earlier than e^k and arriving at d^k no later than l^k. This occurs if and only if:

1. there exists a corresponding arc in the flat network, that is $(n_1, n_2) \in A$,

2. there exists a k-feasible path that uses this arc, i.e., a path from o^k to d^k for which

$$e^k + \gamma\left(o^k, n_1\right) + \tau_{(n_1,n_2)} + \gamma\left(n_2, d^k\right) \leq l^k,$$

3. the intervals $[t_1, t'_1)$ and $[t_2, t'_2)$ contain (or come after) the earliest time that commodity k can visit n_1 and n_2, respectively, using a shortest path from its origin to n_1 and then arc (n_1, n_2), i.e.,

$$e^k + \gamma\left(o^k, n_1\right) < t'_1, \text{ and}$$
$$e^k + \gamma\left(o^k, n_1\right) + \tau_{(n_1,n_2)} < t'_2,$$

4. the intervals $[t_1, t'_1)$ and $[t_2, t'_2)$ contain (or come before) the latest time that commodity k can visit n_1 and n_2, respectively, using a shortest path from n_2 to its

destination, preceded by arc (n_1, n_2), and still reach the destination, i.e.,

$$t_1 + \tau_{(n_1,n_2)} + \gamma\left(n_2, d^k\right) \leq l^k, \text{ and}$$

$$t_2 + \gamma\left(n_2, d^k\right) \leq l^k, \text{ and}$$

5. there exists a time t in the interval $[t_1, t_1')$ such that travel along arc a departing at t arrives within the interval $[t_2, t_2')$, i.e., there exists $t \in [t_1, t_1')$ such that $t + \tau_{(n_1,n_2)} \in [t_2, t_2')$, or, equivalently,

$$t_1 + \tau_{(n_1,n_2)} < t_2', \text{ and}$$

$$t_2 < \tau_{(n_1,n_2)} + t_1'.$$

An alternative way to describe the above five conditions is as follows. First, observe that the interval

$$I^k_{(n_1,n_2)} := \left[e^k + \gamma(o^k, n_1),\ l^k - \gamma(n_2, d^k) - \tau_{n_1,n_2}\right]$$

is the set of times that it is feasible for commodity k to depart node n_1 if using arc (n_1, n_2) in its path. Now define $I^k_{ai} := I^k_a \cap [\mathcal{T}^{n_1}_{(i)}, \mathcal{T}^{n_1}_{(i+1)})$, and define the notation $I + t = [I_1 + t, I_2 + t)$, where $I = [I_1, I_2)$. Then properties 2-5 are equivalent to asking that $(I^k_{ai} + \tau_a) \cap [t_2, t_2')$ is non-empty, where $t_1 = \mathcal{T}^{n_1}_{(i)}$. That is,

$$\left[e^k + \gamma(o^k, n_1) + \tau_{n_1,n_2},\ l^k - \gamma(n_2, d^k)\right] \cap [t_1 + \tau_{n_1,n_2}, t_1' + \tau_{n_1,n_2}) \cap [t_2, t_2') \neq \emptyset.$$

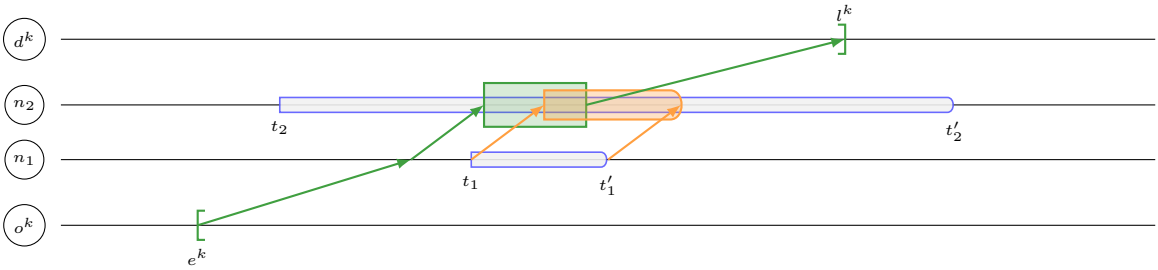


Figure 4.1: Example: Conditions for dispatch arc

We thus define $\mathcal{D}^k_{\mathcal{T}}$ to be the set of timed-arcs for which the above conditions, 1-5, hold.

4.3.2.2 *Holding-arcs*

The holding-arc $a = (i_1, i_2) \in \mathcal{N}_\mathcal{T} \times \mathcal{N}_\mathcal{T}$, connecting two node-intervals $i_1 = (n, t, t')$ and $i_2 = (n, t', t'')$, for commodity k, is included in $\mathcal{H}^k_\mathcal{T}$, if there exists a k-feasible path including node n that can be traversed so that, if $n \neq d^k$, departure from n is at some time in the interval $[t', l^k]$, and, if $n \neq o^k$, arrival at n is at some time in the interval $[e^k, t')$, while still departing from its origin node no earlier than e^k and arriving at its destination node no later than l^k. This occurs if and only if:

1. there exists a k-feasible path that uses node n, i.e., a path from o^k to d^k for which
$$e^k + \gamma(o^k, n) + \gamma(n, d^k) \leq l^k,$$
2. the interval $[t, t')$ contains (or comes after) the earliest time that the commodity can visit node n
$$e^k + \gamma(o^k, n) < t', \text{ and}$$
3. the interval $[t', t'')$ contains (or comes before) the latest time that the commodity k can visit node n and still reach its destination, i.e.,
$$t' + \gamma(n, d^k) \leq l^k.$$

We thus define $\mathcal{H}^k_\mathcal{T}$ to be the set of elements $((n, t, t'), (n, t', t'')) \in \mathcal{N}_\mathcal{T} \times \mathcal{N}_\mathcal{T}$ for which the above conditions, 1-3, hold.

4.4 The Structure of Solutions

The time-expanded network has been defined such that under certain conditions travel between nodes is allowed to be quicker than the given transit times, this admits consolidations that may not be possible in reality, but can reduce the total cost. As a result it is intuitive to see that CTSNDP($\mathcal{T}$) is a relaxation of CTSNDP, for any set of time points $\mathcal{T}$. Moreover, when assuming integer input, a full discretization, that is $\mathcal{T}^n = \{0, 1, \ldots, H\}$

for all $n \in N$, guarantees that every feasible solution to CTSNDP($\mathcal{T}$) is also feasible for CTSNDP.

Our algorithm solves CTSNDP by iteratively solving CTSNDP($\mathcal{T}$) instances and eliminating continuous-time infeasible solutions by adding specific time points to $\mathcal{T}$, or by seeing if a solution can be made feasible without changing the cost. Suitable time points are chosen by exploiting the structure of solutions at each iteration. The following defines the terminology for characterizing a solution.

4.4.1 Paths

Let W^k be the set of all flat-paths, and $\mathcal{W}^k_{\mathcal{T}}$ the set of all timed-paths, for commodity $k \in K$, such that,

$$W^k = \left\{ P^k \in 2^A : \ P^k \text{ is a flat-path from } o^k \text{ to } d^k \right\},$$

$$\mathcal{W}^k_{\mathcal{T}} = \left\{ \mathcal{P}^k_{\mathcal{T}} \in 2^{\mathcal{A}^k_{\mathcal{T}}} : \ \mathcal{P}^k_{\mathcal{T}} \text{ is a timed-path from } \left(o^k, t_o, t'_o\right) \text{ to } \left(d^k, t_d, t'_d\right) \right\},$$

where $t_o \in \mathcal{T}^{o^k}$ is the unique time point such that $t_o \leq e^k < t'_o$, and $t_d \in \mathcal{T}^{d^k}$ is the unique time point such that $t_d \leq l^k < t'_d$. A **path-solution**, $Q^K \in \prod_{k \in K} W^k$, is a system of flat-paths, one for each commodity. Let $Q^k = \text{proj}_k Q^K$ denote the flat-path in Q^K for each commodity $k \in K$. Similarly, a **timed-path-solution**, $\mathcal{Q}^K_{\mathcal{T}} \in \prod_{k \in K} \mathcal{W}^k_{\mathcal{T}}$, is a system of timed-paths, one for each commodity. Let $\mathcal{Q}^k_{\mathcal{T}} = \text{proj}_k \mathcal{Q}^K_{\mathcal{T}}$ denote the timed-path in $\mathcal{Q}^K_{\mathcal{T}}$ for each commodity $k \in K$. Note that Q^K can be defined as a projection of $\mathcal{Q}^K_{\mathcal{T}}$, that is, by removing the time components.

4.4.2 Consolidations

The set of commodities that use timed-arc $a \in \mathcal{D}^K_{\mathcal{T}}$, in the timed-path-solution $\mathcal{Q}^K_{\mathcal{T}}$, is given by the function

$$\kappa_a(\mathcal{Q}^K_{\mathcal{T}}) = \{k \in K \ : \ a \in \mathcal{Q}^k_{\mathcal{T}}\}.$$

These commodities are said to *consolidate* together. In reality, sharing a dispatch-arc does not necessarily imply consolidation. For example consider two commodities each with one

truck load of quantity each, departing at the same time.

A **consolidation** is defined by an arc and a non-empty set of commodities, that is, $(a, \kappa) \in A \times 2^K$. The intention of this consolidation is that every commodity in $\kappa \subseteq K$ shares the same dispatch-arc in their timed-path, corresponding to arc $a \in A$. A set of consolidations can be constructed from the timed-path-solution, $\mathcal{Q}_{\mathcal{T}}^K$, using the function

$$\mathcal{C}(\mathcal{Q}_{\mathcal{T}}^K) = \left\{ \left((n_1, n_2), \kappa_a \left(\mathcal{Q}_{\mathcal{T}}^K \right) \right) \; : \; a = ((n_1, t_1, t_1'), (n_2, t_2, t_2')) \in \bigcup_{k \in K} \mathcal{Q}_{\mathcal{T}}^k, \; n_1 \neq n_2 \right\}.$$

Note that trivial consolidations, i.e. $(a, \kappa) \in \mathcal{C}(\mathcal{Q}_{\mathcal{T}}^K)$ with $|\kappa| = 1$, are typically ignored.

4.4.3 Flat-Solutions

Solutions to CTSNDP($\mathcal{T}$) are known as *timed-solutions*, and, if feasible, have a corresponding timed-path-solution $\mathcal{Q}_{\mathcal{T}}^K$. A **flat-solution** is defined as the pair of a path-solution and a set of consolidations, i.e. $S = (Q^K, C)$. Observe that there are only a finite number of unique flat-solutions, and for each flat-solution there are possibly infinite corresponding timed-solutions.

The flat-solution $S = (Q^K, C)$, is called **representable** in $\mathcal{G}_{\mathcal{T}}^K$ if there exists a feasible solution to CTSNDP($\mathcal{T}$), i.e. $\mathcal{Q}_{\mathcal{T}}^K$, that can be mapped to S. This mapping has Q^K as the projection of $\mathcal{Q}_{\mathcal{T}}^K$, and $C = \mathcal{C}(\mathcal{Q}_{\mathcal{T}}^K)$. Similarly, S is called **implementable** if there exists a feasible solution to CTSNDP that can be mapped to S. Note that implementability implies representability, but not the converse.

Therefore, the flat-solution S is implementable if and only if there exists departure time t_n^k (or arrival time if $n = d^k$), for every commodity $k \in K$, and each $n \in Q^k$, such that (4.2), below, holds. In order to achieve consolidation $((n_1, n_2), \kappa) \in C$, we require $t_{n_1}^{k_1} = t_{n_1}^{k_2}$ for all $k_1, k_2 \in \kappa$; this can be modeled with the introduction of an auxiliary variable, s_{n_1,n_2}^{κ}, which represents the departure time of the consolidating commodities κ

along arc (n_1, n_2), as follows:

$$\begin{aligned} t^k_{o^k} &\geq e^k, && \forall\, k \in K, \\ t^k_{n_2} &\geq t^k_{n_1} + \tau_{(n_1,n_2)}, && \forall\, k \in K,\ (n_1, n_2) \in Q^k, \\ t^k_{d^k} &\leq l^k, && \forall\, k \in K, \text{ and} \\ t^k_{n_1} &= s^\kappa_{n_1,n_2}, && \forall\, ((n_1, n_2),\ \kappa) \in C,\ k \in \kappa. \end{aligned} \tag{4.2}$$

The set of times, $(t^k_n)^{k\in K}_{n\in N}$, or (t^k_n) for short, satisfying (4.2) is referred to as a **corresponding timing**.

4.4.4 Solution Graph

Visualizing a flat-solution as a graph is an intuitive way to see the interactions and flow of commodities in the form of consolidations. Furthermore, as will be shown later, it is also instrumental in the check for implementability, and the choice for refining the discretization. Let $GS(S) = (\mathcal{V}, \mathcal{E})$ be a directed graph constructed from the flat-solution $S = (Q^K, C)$, where the nodes $\mathcal{V}$ are the union of commodity dispatch nodes, $\mathcal{V}^t = \{t^k_n :\ k \in K,\ n \in Q^k\}$, and consolidation nodes, $\mathcal{V}^s = \{s^\kappa_{n_1,n_2}\}_{((n_1,n_2),\kappa)\in C}$; and edges $\mathcal{E}$ are defined by:

$$\begin{aligned} \mathcal{E} = &\{(t^k_{n_1}, s^\kappa_{n_1,n_2}) :\ s^\kappa_{n_1,n_2} \in \mathcal{V}^s,\ k \in \kappa\} \\ &\cup \{(s^\kappa_{n_1,n_2}, t^k_{n_2}) :\ s^\kappa_{n_1,n_2} \in \mathcal{V}^s,\ k \in \kappa\} \\ &\cup \{(t^k_{n_1}, t^k_{n_2}) :\ k \in K,\ (n_1, n_2) \in Q^k,\ \nexists\, s^\kappa_{n_1,n_2} \in \mathcal{V}^s \text{ with } k \in \kappa\}. \end{aligned}$$

The edge lengths, referred to as transit times, are defined by $\rho_{(v_1,v_2)}$ for all $(v_1, v_2) \in \mathcal{E}$, where

$$\rho_{(v_1,v_2)} = \begin{cases} \tau_{(n_1,n_2)} & \text{if } (v_1, v_2) = (t^k_{n_1}, t^k_{n_2}) \text{ for some } k \in K, (n_1, n_2) \in Q^k \\ \tau_{(n_1,n_2)} & \text{if } (v_1, v_2) = (s^\kappa_{n_1,n_2}, t^k_{n_2}) \text{ for some } ((n_1, n_2), \kappa) \in C, k \in \kappa \\ 0 & \text{o/w} \end{cases}$$

Observe that these nodes, edges, and edge lengths are a partial representation of the inequalities defining implementability in (4.2). To illustrate these definitions, consider the following example:

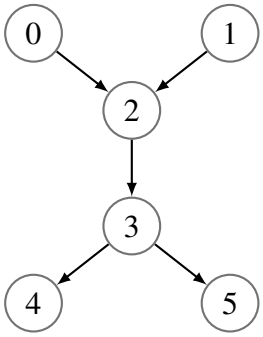


Figure 4.2: Network

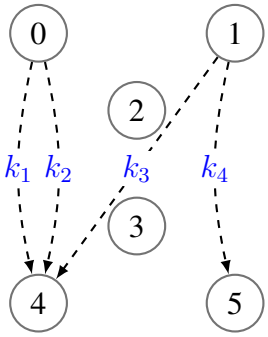


Figure 4.3: Commodities

k	Q^k	(n_1, n_2)	κ
k_1	0,2,3,4	$0, 2$	$\{k_1, k_2\}$
k_2	0,2,3,4	$2, 3$	$\{k_1, k_4\}$
k_3	1,2,3,4	$2, 3$	$\{k_2, k_3\}$
k_4	1,2,3,5	$3, 4$	$\{k_1, k_2, k_3\}$

Figure 4.4: Example $S = (Q^K, C)$

Which gives the solution graph:

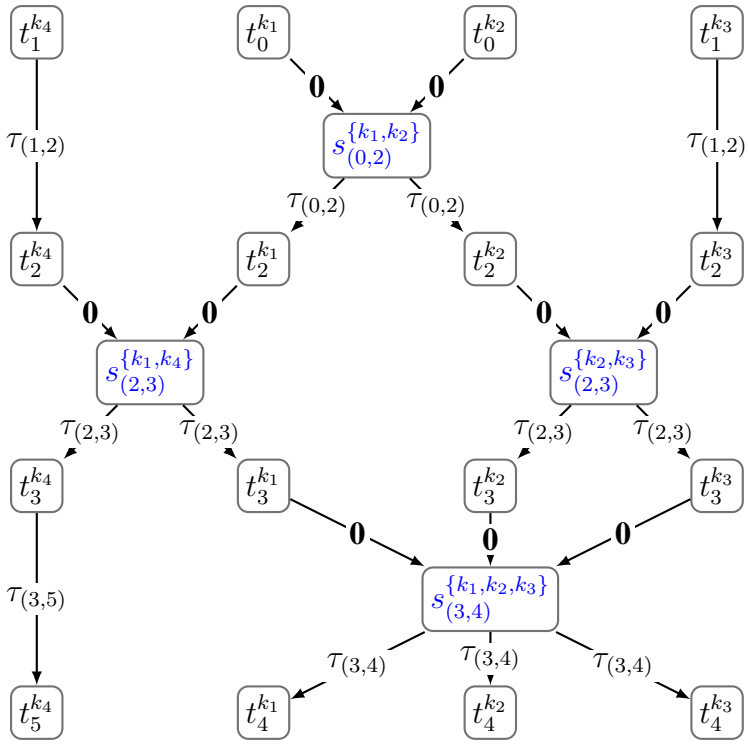


Figure 4.5: Solution Graph $GS(S)$ Example

4.4.4.1 *Implementability*

To see how the solution graph can be used to check for implementability, first consider that the system of inequalities in (4.2) can be rearranged to give

$$\begin{aligned}
-t^k_{o^k} &\leq -e^k, && \forall\, k \in K, \\
t^k_{n_1} - t^k_{n_2} &\leq -\tau_{(n_1,n_2)}, && \forall\, k \in K,\ (n_1,n_2) \in Q^k, \\
t^k_{d^k} &\leq l^k, && \forall\, k \in K, \\
t^k_{n_1} - s^\kappa_{(n_1,n_2)} &\leq 0, && \forall\, ((n_1,n_2),\ \kappa) \in C,\ k \in \kappa, \text{ and} \\
s^\kappa_{(n_1,n_2)} - t^k_{n_1} &\leq 0, && \forall\, ((n_1,n_2),\ \kappa) \in C,\ k \in \kappa.
\end{aligned} \tag{4.3}$$

Which can be written as a system of difference constraints by adding dummy variables D, D':

$$\begin{aligned}
D' - D &\leq 0, \\
D - t^k_{o^k} &\leq -e^k, && \forall\, k \in K, \\
t^k_{n_1} - t^k_{n_2} &\leq -\tau_{(n_1,n_2)}, && \forall\, k \in K,\ (n_1,n_2) \in Q^k, \\
t^k_{d^k} - D' &\leq l^k, && \forall\, k \in K, \\
t^k_{n_1} - s^\kappa_{(n_1,n_2)} &\leq 0, && \forall\, ((n_1,n_2),\ \kappa) \in C,\ k \in \kappa, \text{ and} \\
s^\kappa_{(n_1,n_2)} - t^k_{n_1} &\leq 0, && \forall\, ((n_1,n_2),\ \kappa) \in C,\ k \in \kappa.
\end{aligned} \tag{4.4}$$

Thus, by extending the solution graph $GS(S) = (\mathcal{V}, \mathcal{E})$ to include dummy nodes D, D' and arcs corresponding to (4.4), the constraint graph $\widehat{GS}(S) = (\widehat{\mathcal{V}}, \widehat{\mathcal{E}})$ is defined such that:

$$\begin{aligned}
\widehat{\mathcal{V}} &= \mathcal{V} \cup \{D, D'\}, \\
\widehat{\mathcal{E}} &= \mathcal{E} \cup \{(D, D')\} \\
&\quad \cup \{(D, t^k_{o^k}) \colon k \in K\} \\
&\quad \cup \{(t^k_{d^k}, D') \colon k \in K\} \\
&\quad \cup \{(s^\kappa_{n_1,n_2}, t^k_{n_1}) : (t^k_{n_1}, s^\kappa_{n_1,n_2}) \in \mathcal{E}\},
\end{aligned}$$

and edge length (as transit time), for all $(v_1, v_2) \in \widehat{\mathcal{E}}$ having

$$\widehat{\rho}_{(v_1,v_2)} = \begin{cases} e^k & \text{if } (v_1, v_2) = (D, t^k_{o^k}) \text{ for some } k \in K \\ -l^k & \text{if } (v_1, v_2) = (t^k_{d^k}, D') \text{ for some } k \in K \\ 0 & \text{if } (v_1, v_2) = (D, D') \\ \rho_{(v_1,v_2)} & \text{o/w} \end{cases}$$

Observe that $\widehat{GS}(S)$ is the graph representation of the inequalities in (4.4), and that $GS(S)$ is a subgraph of $\widehat{GS}(S)$. To illustrate these extensions, Figure 4.6 shows the constraint graph corresponding to the example solution graph in Figure 4.5, with the additional nodes and edges highlighted for comparison.

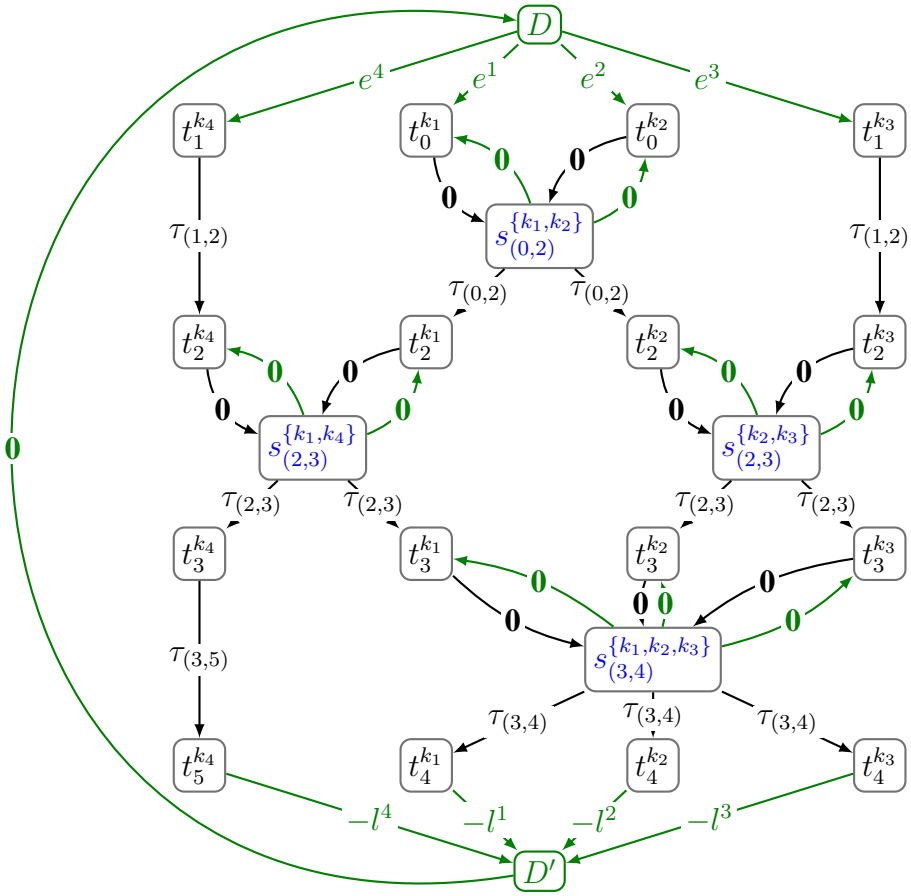


Figure 4.6: Constraint Graph $\widehat{GS}(S)$ Example

Proposition 3. *The flat-solution $S = (Q^K, C)$, is implementable if and only if there does not exist a positive length cycle in the constraint graph $\widehat{GS}(S)$.*

Proof. From known results (Ahuja, Magnanti, and Orlin, [1993], pp. 103-104) the inequalities (4.4) are feasible, if and only if the constraint graph $\widehat{GS}(S)$ does not have a positive length cycle. By definition, the flat-solution S is implementable if and only if (4.3) is feasible. Hence it is sufficient to show that the inequalities (4.3) are equivalent to (4.4).

Observe that (4.4) is simply (4.3) shifted by D: for (4.3) $\Rightarrow$ (4.4) simply take $D = D' = 0$. For (4.3) $\Leftarrow$ (4.4), let $\{(\hat{t}^k_n), D, D'\}$ be a solution of (4.4), and let $t^k_n = \hat{t}^k_n - D$. Then clearly (t^k_n) satisfies (4.3), and the only constraint that does not follow immediately is given by

$$\begin{aligned} \hat{t}^k_{d^k} - D' &\leq l^k \\ t^k_{d^k} + D - D' &\leq l^k \\ t^k_{d^k} &\leq l^k + D' - D, \end{aligned}$$

and $D' - D \leq 0$ implies $t^k_{d^k} \leq l^k$. □

4.4.4.2 Notation for Common Graph Functions

Since the solution and constraint graphs have a significant role in the DDDI algorithm, the following definitions are used to simplify the notation.

Let $\widetilde{P} = (\tilde{n}_1, \tilde{n}_2, \ldots, \tilde{n}_r)$ be a simple path on the network $\widetilde{G} = (\widetilde{N}, \widetilde{A})$. For node $n \in \widetilde{P}$ we write $\widetilde{P}^{\to n}$ to denote the subpath of $\widetilde{P}$ given by $(\tilde{n}_1, \ldots, \tilde{n}_i)$, where i is the unique index, $i \in \{1, \ldots, r\}$, such that $\tilde{n}_i = n$. Now, extend this notation to handle simple cycles: let the simple cycle $\mathring{P}$ be specified as $\mathring{P} = (n_1, n_2, \ldots, n_r, n_1)$, we define $\mathring{P}^{\to n_1}$ to be (n_1), that is, the first occurrence of n_1. Since the remaining nodes $n \in \mathring{P} \setminus \{n_1\}$ are uniquely indexed, the function is (already) well defined.

For any function of the form $\psi\colon \widetilde{A} \to \mathbb{R}$, we write

$$\psi(\widetilde{P}) = \sum_{a \in \widetilde{P}} \psi(a),$$

or similarly, for any vector $\psi \in \mathbb{R}^{|\tilde{A}|}$, we write

$$\psi(\widetilde{P}) = \sum_{a \in \widetilde{P}} \psi_a.$$

As an example, given flat-solution S, let P be a simple path in the solution graph $GS(S)$. Then, the total transit time for that path is given by $\rho(P)$, and for node $t_n^k \in P$, $\rho(P^{\to t_n^k})$ gives the total transit time of the path up to and including node t_n^k.

4.4.5 Implementability

Using the results above, a check for implementability for the flat-solution S is reduced to a check for positive length cycles on the corresponding constraint graph, which can be done efficiently, in principle, by applying a label-correcting shortest path algorithm (with cycle detection) to the network. In practice, it is possible to exploit special structure in the network, allowing the positive length cycles to be detected more efficiently via a two-step process, in which the second step is a shortest path algorithm in an acyclic graph. In addition the information calculated in the second step can be used to determine how to refine the discretization. The special structure is as follows.

Observe that, from the construction of the constraint graph, it is obvious that a cycle in the constraint graph must take one of three forms:

(i) it consists of the pair of zero-length arcs corresponding to solution graph arcs of the form $(t_n^k, s_{n,n'}^\kappa)$,

(ii) it corresponds to a cycle in the solution graph, or

(iii) the cycle uses (D', D).

The first form cannot have positive length. Any cycle in the constraint graph that corresponds to a cycle in the solution graph is of the second form and must have positive cost. This gives the first main result.

Lemma 5. *If there is a cycle in the solution graph $GS(S)$, then* (4.2) *has no solution, and hence S is non-implementable.*

Finally then, consider the case when the solution graph is acyclic, and any cycle in the constraint graph must have the form (D, P, D'), where P is a path (possibly including zero-length loops around pairs of the form $(t^k_n, s^\kappa_{n,n'})$ in the constraint graph from node $t^k_{o^k}$ to node $t^{k'}_{d^{k'}}$ for k and k' two (possibly identical) commodities.

Since a positive length cycle exists in a graph if and only if there exists a *simple* positive length cycle, the flat-solution S is implementable if and only if there is no simple positive length cycle in the constraint graph, which (in this case) holds if and only if there is no positive length cycle in the constraint graph of the form (D, P, D'), where P is a simple path in the constraint graph from $t^k_{o^k}$ to $t^{k'}_{d^{k'}}$ for some k, k'. This holds if and only if there is no path P' in the solution graph from $t^k_{o^k}$ to $t^{k'}_{d^{k'}}$ with $e^k + \rho(P') - l^{k'} > 0$. This gives the second main result.

Lemma 6. *If the solution graph is acyclic, then* (4.2) *has no solution, and hence S is non-implementable, if and only if there exists commodities $k, k' \in K$, (possibly identical), and a simple path P, from $t^k_{o^k}$ to $t^{k'}_{d^{k'}}$, in the solution graph with $e^k + \rho(P) > l^{k'}$.*

4.4.6 Earliest Departure Time

Given the solution $S = (Q^K, C)$, with acyclic solution graph $GS(S) = (\mathcal{V}, \mathcal{E})$, let the earliest departure time E_v be defined for a node $v \in \mathcal{V}$ such that:

$$E_v = \begin{cases} e^k & \text{if } v = t^k_{o^k} \text{ for some } k \in K \\ \max\limits_{k \in \kappa}(E_{t^k_{n_1}}) & \text{if } v = s^\kappa_{n_1,n_2} \text{ for some } ((n_1,n_2),\kappa) \in C \\ E_{v_1} + \rho(v_1, v) & \text{if } v = t^k_n, n \neq o^k, \text{and } (v_1, v) \in \mathcal{E} \text{ for some } k \in K \end{cases}$$

Note that E_v is well defined due to the implicit structure of the solution graph, that is, if $v \in \mathcal{V}$ has $v = t^k_n$, $n \neq o^k$, for some k and n, then there must exist a unique arc of the form $(v_1, v) \in \mathcal{E}$. Thus we have:

Lemma 7. *For the solution $S = (Q^K, C)$ with acyclic solution graph, the length of the shortest path from D to v in the constraint graph is E_v.*

4.5 Refining the Discretization

Consider all the feasible solutions to CTSNDP($\mathcal{T}$), with discretization $\mathcal{T}$. Recall that there are only a finite number of flat-solutions corresponding to all these feasible solutions, and at least one of these is implementable (assuming that a feasible continuous-time solution exists, i.e. $e^k + \gamma(o^k, d^k) \leq l^k$ for all $k \in K$). A discretization is *refined* by adding time points, that is, $\mathcal{T}'$ is a refinement of $\mathcal{T}$ if $\mathcal{T} \subset \mathcal{T}'$. Suppose that the flat-solution S, corresponding to a feasible solution of CTSNDP($\mathcal{T}$), is not implementable. It would be nice to choose a refinement $\mathcal{T}'$ such that S does not correspond to any feasible solutions to CTSNDP($\mathcal{T}'$). This is the goal of refining the discretization; to strictly reduce the number of non-implementable flat-solutions, typically targeting a particular flat-solution to "remove". Since it is only possible to remove non-implementable flat-solutions, of which there are finitely many, it is intuitive to see that repeating this process will guarantee that an optimal solution of CTSNDP($\mathcal{T}'$) will be implementable, and hence continuous-time optimal (since it is a lower bound of CTSNDP).

Suitable time points can be determined by exploiting the structure of S. Specifically,

they correspond to particular paths and cycles in the solution graph $GS(S)$. The following lemma and theorems cover these concepts in detail, but first it is important to establish the connection between the solution graph and the time-expanded network induced by the discretization $\mathcal{T}$.

OBSERVATION 2. The solution graph is the network equivalent of a flat-solution, so, if the flat-solution $S = (Q^K, C)$ is representable in $\mathcal{G}^K_\mathcal{T}$, then every path in $GS(S)$, has a corresponding path in $\mathcal{G}^K_\mathcal{T}$. Furthermore, any cycle in $GS(S)$ must also have a corresponding cycle in $\mathcal{G}^K_\mathcal{T}$ – to see this, consider that cycles in the solution graph correspond to consolidations in the timed-solution that include at least one commodity that travels back in time (see Section 4.5.1.3).

OBSERVATION 3. Consider the discretization $\mathcal{T}' \supset \mathcal{T}$, that is, $\mathcal{T}'$ is a refinement of $\mathcal{T}$; if there exists a path in $GS(S)$, that does not have a corresponding path in $\mathcal{G}^K_{\mathcal{T}'}$, then the flat-solution $S = (Q^K, C)$ is not representable in $\mathcal{G}^K_{\mathcal{T}'}$ – that is, no feasible solutions to CTSNDP($\mathcal{T}'$) can correspond to S. This holds similarly for cycles.

These two observations are essential to the theorems below. In addition to these, Lemma 8 gives a lower bound on the dispatch times of commodities along a simple path through the $\mathcal{G}^K_\mathcal{T}$, when $\mathcal{T}$ contains time points of a particular form.

Lemma 8. *Let* $\widehat{P} = ((n_1, t_1, t'_1),\ (n_2, t_2, t'_2),\ \ldots,\ (n_r, t_r, t'_r))$ *be a simple path in* $(\mathcal{N}_\mathcal{T}, \mathcal{A}^K_\mathcal{T})$ *starting at* $n_1 = o^k$ *with* $t_1 \geq e^k$*, for some* $k \in K$*, and suppose*

$$\left(n_i,\ e^k + \tau\left(\widehat{P}^{\rightarrow(n_i,t_i,t'_i)}\right)\right) \in \mathcal{T}, \quad \textit{for all } i = 1, \ldots, r,$$

then

$$t_i \geq e^k + \tau\left(\widehat{P}^{\rightarrow(n_i,t_i,t'_i)}\right), \quad \textit{for all } i = 1, \ldots, r.$$

Proof. Consider the induction hypothesis

$$H(i): \quad t_i \geq e^k + \tau\left(\widehat{P}^{\rightarrow(n_i,t_i,t'_i)}\right), \quad \text{for some } i = 1,\ldots,r-1.$$

It is true by assumption for $i = 1$, and since $\tau(\widehat{P}^{\rightarrow(n_1,t_1,t'_1)}) = 0$. Suppose that $H(i)$ is true for some $i \in \{1,\ldots,r-1\}$. Recall that $\mathcal{A}^K_{\mathcal{T}} = \bigcup_{k\in K} \mathcal{A}^k_{\mathcal{T}}$, then observe that there exists $k' \in K$ such that $((n_i,t_i,t'_i),(n_{i+1},t_{i+1},t'_{i+1})) \in \mathcal{A}^{k'}_{\mathcal{T}}$, so by $[\![5]\!]$ on $\mathcal{A}^{k'}_{\mathcal{T}}$

$$\begin{aligned} t'_{i+1} &> t_i + \tau_{(n_i,n_{i+1})} \\ &\geq e^k + \tau(\widehat{P}^{\rightarrow(n_i,t_i,t'_i)}) + \tau_{(n_i,n_{i+1})} \\ &= e^k + \tau(\widehat{P}^{\rightarrow(n_{i+1},t_{i+1},t'_{i+1})}) \end{aligned}$$

By Proposition 2, and since $(n_{i+1},\ e^k + \tau(\widehat{P}^{\rightarrow(n_{i+1},t_{i+1},t'_{i+1})})) \in \mathcal{T}$ it follows that

$$t_{i+1} \geq e^k + \tau(\widehat{P}^{\rightarrow(n_{i+1},t_{i+1},t'_{i+1})}),$$

and $H(i)$ holds. □

Theorem 4. *Let S be a non-implementable flat-solution that is representable in $\mathcal{G}^K_{\mathcal{T}}$ and has an acyclic solution graph $GS(S)$. Furthermore, let P be a simple path in $GS(S)$ from $t^{k_1}_{o^{k_1}}$ to $t^{k_2}_{d^{k_2}}$ for some $k_1,k_2 \in K$ with $e^{k_1} + \rho(P) > l^{k_2}$, and $e^{k_1} + \rho(P^{\rightarrow t^k_n}) = E_{t^k_n}$, where $E_{t^k_n}$ is the earliest departure time from node $t^k_n \in P$. Then S is not representable in $\mathcal{G}^K_{\mathcal{T}'}$ with discretization refinement*

$$\mathcal{T}' = \mathcal{T} \cup \left\{\left(n, E_{t^k_n}\right) \mid t^k_n \in P \cap \mathcal{V}^t\right\}.$$

Proof. First observe that since the flat-solution S is non-implementable and the solution graph $GS(S)$ is acyclic, the path P, defined in the theorem, exists. Specifically, from Lemma 6 there exists a simple path, P, in $GS(S)$, from $t^{k_1}_{o^{k_1}}$ to $t^{k_2}_{d^{k_2}}$ for some $k_1,k_2 \in K$, such that $e^{k_1} + \rho(P) > l^{k_2}$. Moreover, this path can always be chosen without any holding time, that is, for all t^k_n in P: $E_{t^k_n} = e^{k_1} + \rho(P^{\rightarrow t^k_n})$. To see this, walk up the graph from $t^{k_2}_{d^{k_2}}$ following the $\arg\max_{k\in\kappa} E_{t^k_n}$ commodity at each consolidation.

Suppose that S is representable in $\mathcal{G}^K_{\mathcal{T}'}$, then from Observation 2 there exists the simple

path $\widehat{P}$ in $(\mathcal{N}_{\mathcal{T}'}, \mathcal{A}^K_{\mathcal{T}'})$ that corresponds to P, in particular having

$$a = ((n_{|\widehat{P}|-1}, t_{|\widehat{P}|-1}, t'_{|\widehat{P}|-1}), (d^{k_2}, t_{|\widehat{P}|}, t'_{|\widehat{P}|})) \in \mathcal{A}^{k_2}_{\mathcal{T}'},$$

and with

$$\tau(\widehat{P}^{\to i}) = \rho(P^{\to t^k_n}),$$

for all $i = (n, t, t')$, the node-interval corresponding with a visit to t^k_n. Observe that the path P may visit the node $n \in N$ with multiple commodities at different times, but a single commodity will visit n at most once. Clearly then i has a unique correspondence to t^k_n, and hence

$$E_{t^k_n} = e^{k_1} + \tau(P^{\to i}).$$

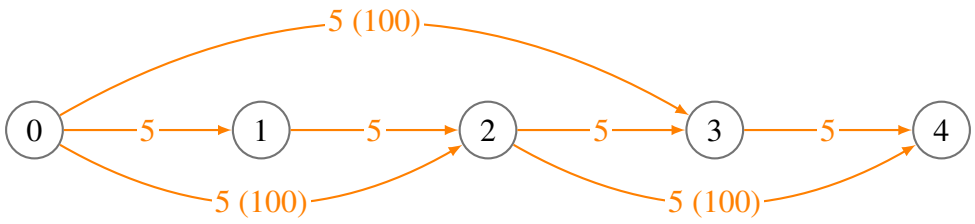
Thus by Lemma 8 on $\widehat{P}$,

$$t_{|\widehat{P}|} \geq e^{k_1} + \tau(\widehat{P}) = e^{k_1} + \rho(P) > l^{k_2},$$

but this violates $[\![4]\!]$ on $\mathcal{A}^{k_2}_{\mathcal{T}'}$. Thus $a \notin \mathcal{A}^{k_2}_{\mathcal{T}'}$, and there does not exist path $\widehat{P}$ corresponding to P, hence by Observation 3, S is not representable in $\mathcal{G}^K_{\mathcal{T}'}$. □

Suppose that the solution to CTSNDP($\mathcal{T}$) yields the non-implementable flat-solution S. Theorem 4 shows that there is a path in the solution graph $GS(S)$ (which can be easily found by a depth-first search on an acyclic graph), such that adding the time points along this path sufficiently refines the discretization to "remove" S.

Note that the time points used in Theorem 4 are not unique, nor minimal. To see this, consider the example below in which a single commodity, k, travels from 0 to 4 with $e^k = 0$, $l^k = 17$. Travel times and costs (in brackets when different to travel time) are listed on the arcs. Let the initial discretization be $\mathcal{T} = N \times \{0, 20\}$.



The path $(0, 1, 2, 3, 4)$ is an optimal solution to CTSNDP($\mathcal{T}$), but it is not k-feasible.

Using Theorem 4,

$$\mathcal{T}_1' = \mathcal{T} \cup \{(1,5),(2,10),(3,15),(4,20)\},$$

however it can be shown that

$$\mathcal{T}_2' = \mathcal{T} \cup \{(2,t)\}, \text{ for any } t \in (7,10],$$

is also a suitable refinement. To see that this single time point is sufficient, see the figures and proof below.

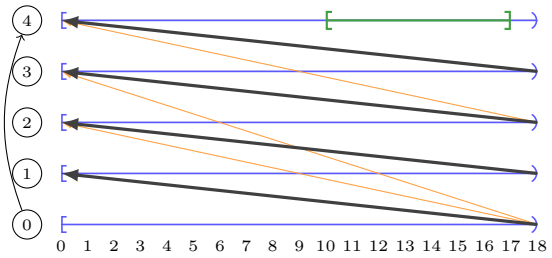


Figure 4.7: Timed-arcs and Solution on $\mathcal{T}$

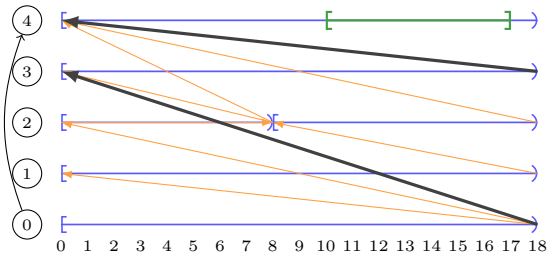


Figure 4.8: Timed-arcs and Solution on $\mathcal{T}_2'$

Consider the time-arcs $((1,t_1,t_1'),(2,t,t'))$, and $((2,t,t'),(3,t_3,t_3'))$. Notice that they share the same node-interval $(2,t,t')$. From the conditions for time-arcs, observe that t is valid for $[\![3]\!]$ on $\mathcal{A}_{\mathcal{T}}^k$ when:

$$e^k + \gamma(o^k, n_1) < t_2' - \tau_{(n_1,n_2)}$$

$$0 + \gamma(0,2) < t' - \tau_{(2,3)}$$

$$t' > 10.$$

and from $[\![4]\!]$ on $\mathcal{A}_{\mathcal{T}}^k$ when:

$$t_1 + \tau_{(n_1,n_2)} + \gamma(n_2, d^k) \leq l^k$$

$$t + \tau_{(2,3)} + \gamma(3,4) \leq 17$$

$$t \leq 7,$$

Adding the time point $(2,t)$ creates the node-intervals $(2,0,t)$, and $(2,t,20)$. If t is chosen to be within $(7,10]$, this ensures that neither node-interval is valid for both inequalities (and hence an associated time-arc) above. Since no timed-arc can connect these node-intervals, the timed-path cannot exist, and by Observation 3 the discretization

has been sufficiently refined.

With this in mind, there can be many alternate discretization refinement approaches, and the choice can have a significant impact on the size of the discretization and speed of convergence. The next section considers a few approaches for special case scenarios.

4.5.1 Special Cases

4.5.1.1 Path

Suppose that for commodity k, the length of its path in the flat-solution is too long. Then, under certain circumstances (Theorem 5), it is possible to refine the discretization using a single *unary* time point. Unary time points are almost always preferred, as they effectively refine the discretization, without dramatically increasing the corresponding time-expanded network.

Theorem 5. *If the non-implementable flat-solution $S = (Q^K, C)$, is representable in $\mathcal{G}^K_{\mathcal{T}}$ and there exists $(n_1, n_2), (n_2, n_3) \in Q^k$ such that*

$$e^k + \gamma(o^k, n_1) + \tau_{(n_1,n_2)} + \tau_{(n_2,n_3)} + \gamma(n_3, d^k) > l^k,$$

for some $k \in K$, then S is not representable in $\mathcal{G}^K_{\mathcal{T}'}$, where

$$\mathcal{T}' = \mathcal{T} \cup \{(n_2,\, e^k + \gamma(o^k, n_1) + \tau_{(n_1,n_2)})\}.$$

Proof. Suppose that S is representable in $\mathcal{G}^K_{\mathcal{T}'}$, then by Observation 2 there exists a timed-path in $\mathcal{G}^k_{\mathcal{T}'}$ corresponding to the flat-path Q^k; more specifically, there exist timed-arcs

$$((n_1, t_1, t_1'), (n_2, t_2, t_2')),\ ((n_2, \hat{t}_2, \hat{t}_2'), (n_3, t_3, t_3')) \in \mathcal{A}^k_{\mathcal{T}'},$$

with $t_2 \leq \hat{t}_2$. Then using Proposition 2, with $(n_2,\, e^k + \gamma(o^k, n_1) + \tau_{(n_1,n_2)}) \in \mathcal{T}'$, and $[\![3]\!]$ on $\mathcal{A}^k_{\mathcal{T}'}$ implies

$$e^k + \gamma(o^k, n_1) + \tau_{(n_1,n_2)} \leq t_2,$$

and $[\![4]\!]$ on $\mathcal{A}^k_{\mathcal{T}'}$ requires

$$\hat{t}_2 + \tau_{(n_2,n_3)} + \gamma(n_3, d^k) \leq l^k,$$

but this implies

$$e^k + \gamma(o^k, n_1) + \tau_{(n_1,n_2)} + \tau_{(n_2,n_3)} + \gamma(n_3, d^k) \leq l^k.$$

This contradicts the assumption, so at least one of the above timed-arcs cannot exist. Thus the timed-path cannot exist, and by Observation 3, S is not representable in $\mathcal{G}^K_{\mathcal{T}'}$. $\square$

4.5.1.2 Disjoint Time-Window

A *time-window* for node n and commodity k, defines the earliest and latest times that k could visit n. Suppose that in a flat-solution, at least two commodities k_1, and k_2 consolidate over (n_1, n_2), such that in reality the earliest time k_1 can reach n_2 (traveling via n_1) is after the latest time that k_2 can leave and still reach its destination in time. In this scenario, commodities k_1, k_2 have disjoint time-windows, and under certain conditions (Theorem 6) it is possible to refine the discretization with a unary time point.

Theorem 6. *If the non-implementable flat-solution $S = (Q^K, C)$, is representable in $\mathcal{G}^K_{\mathcal{T}}$ and there exists (n_1, n_2) in both Q^{k_1} and Q^{k_2} such that*

$$e^{k_1} + \gamma(o^{k_1}, n_1) + \tau_{(n_1,n_2)} + \gamma(n_2, d^{k_2}) > l^{k_2},$$

for some $k_1, k_2 \in K$, then S is not representable in $\mathcal{G}^K_{\mathcal{T}'}$, where

$$\mathcal{T}' = \mathcal{T} \cup \{(n_2,\, e^{k_1} + \gamma(o^{k_1}, n_1) + \tau_{(n_1,n_2)})\}.$$

Proof. Suppose that S is representable in $\mathcal{G}^K_{\mathcal{T}'}$, then by Observation 2 there exists timed-paths in $\mathcal{G}^K_{\mathcal{T}'}$ corresponding to Q^{k_1}, and Q^{k_2}; more specifically, there exist timed-arc $a = ((n_1, t_1, t'_1), (n_2, t_2, t'_2))$, such that $a \in \mathcal{A}^{k_1}_{\mathcal{T}'}$, and $a \in \mathcal{A}^{k_2}_{\mathcal{T}'}$. Observe that $[\![4]\!]$ on $\mathcal{A}^{k_2}_{\mathcal{T}'}$ requires

$$t_2 + \gamma(n_2, d^{k_2}) \leq l^{k_2},$$

and $[\![3]\!]$ on $\mathcal{A}^{k_1}_{\mathcal{T}'}$ requires

$$e^{k_1} + \gamma(o^{k_1}, n_1) + \tau_{(n_1,n_2)} < t'_2.$$

By assumption $(n_2,\ e^{k_1} + \gamma(o^{k_1}, n_1) + \tau_{(n_1,n_2)}) \in \mathcal{T}'$, so Proposition 2 gives

$$e^{k_1} + \gamma(o^{k_1}, n_1) + \tau_{(n_1,n_2)} \leq t_2,$$

but this implies that

$$e^{k_1} + \gamma(o^{k_1}, n_1) + \tau_{(n_1,n_2)} + \gamma(n_2, d^{k_2}) \leq l^{k_2}.$$

Since this contradicts assumptions, then $a \notin \mathcal{A}^{k_1}_{\mathcal{T}'}$, or $a \notin \mathcal{A}^{k_2}_{\mathcal{T}'}$, and hence the timed-path cannot exist, and by Observation 3, S is not a representable solution in $\mathcal{G}^K_{\mathcal{T}'}$. □

4.5.1.3 Cycle

Theorem 4 assumes that the solution graph is acyclic, however this is not always the case. If there exists a cycle, then Theorem 7 shows how to sufficiently refine the discretization. The structure of the time points chosen in Theorem 7 is similar to those in chosen in Theorem 4, however handling cycles typically requires significantly more time points. Observe that Theorems 5 and 6 do not make the acyclic assumption, and so even if a cycle is detected it may be possible (and preferable) to use those unary time point theorems. To see how a cycle could arise, consider the following example:

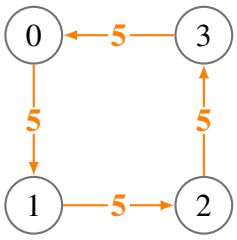


Figure 4.9: Network

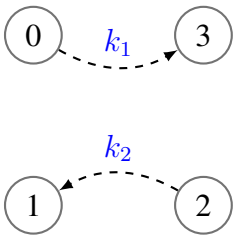


Figure 4.10: Commodities

k	Q^k	(n_1, n_2)	κ
k_1	0,1,2,3	$0, 1$	$\{k_1, k_2\}$
k_2	2,3,0,1	$2, 3$	$\{k_1, k_2\}$

Figure 4.11: Cycle $S = (Q^K, C)$

Which gives the solution graph:

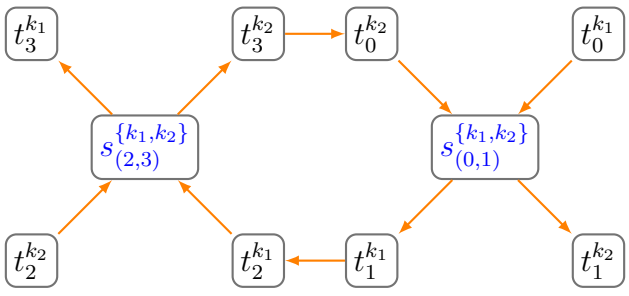


Figure 4.12: Example Cycle Instance

Theorem 7. *Let S be a non-implementable flat-solution that is representable in $\mathcal{G}^K_{\mathcal{T}}$, with solution graph $GS(S)$ containing the simple cycle $\mathring{P}$. Furthermore, let P be a shortest path in $GS(S)$ from $t^{k_1}_{o^{k_1}}$ to $\mathring{P}$ for some $k_1 \in K$, let $\{v\} = P \cap \mathring{P}$, define the cycle such that $\mathring{P} = (v, \mathring{P}_{(2)}, \ldots, \mathring{P}_{(|\mathring{P}|)}, v)$, and let $\mathring{m} = \min\{m : e^{k_1} + \rho(P) + m\rho(\mathring{P}) > l^{k_1}\}$. Then S is not representable in $\mathcal{G}^K_{\mathcal{T}'}$ with refinement:*

$$\mathcal{T}' = \mathcal{T} \cup \left\{ \left(n, e^{k_1} + \rho\left(P^{\to t^k_n}\right)\right) \mid t^k_n \in P \cap \mathcal{V}^t \right\}$$
$$\cup \left\{ \left(n, e^{k_1} + \rho\left(P\right) + m\,\rho(\mathring{P}) + \rho\left(\mathring{P}^{\to t^k_n}\right)\right) \mid t^k_n \in \mathring{P} \cap \mathcal{V}^t,\, m = 0, \ldots, \mathring{m} \right\}.$$

Proof. Suppose S is representable in $\mathcal{G}^K_{\mathcal{T}'}$, then from Observation 2 there exists the simple path $\widehat{P}$ and simple cycle $\mathring{\widehat{P}}$ in $(\mathcal{N}_{\mathcal{T}'}, \mathcal{A}^K_{\mathcal{T}'})$ that corresponds to P and $\mathring{P}$ respectfully, moreover, having $\widehat{P} \cap \mathring{\widehat{P}} = \{\mathring{\widehat{P}}_{(1)}\} \subset \mathcal{N}_{\mathcal{T}'}$.

Observe that $\tau(\widehat{P}^{\to i}) = \rho(P^{\to t^k_n})$ for all $i = (n, t, t')$, that is, the node-interval in $\widehat{P}$ uniquely corresponding with a visit to t^k_n, so by Lemma 8 on $\widehat{P}$:

$$t_{|\widehat{P}|} \geq e^{k_1} + \tau(\widehat{P}) = e^{k_1} + \rho(P).$$

Let $r = |\mathring{P}|$, let $n_1, n_r \in N$ be the nodes visited by $\mathring{P}_{(1)}$ and $\mathring{P}_{(r)}$ respectfully, and let $i^0_r,\, i^1_1 \in \mathcal{N}_{\mathcal{T}'}$ be the node-interval corresponding to the first visit to n_r, and the first return to n_1 in $\mathring{\widehat{P}}$ respectfully. Observe that joining P and $\mathring{P}^{\to \mathring{P}_{(r)}}$ yields a simple path, and so similarly by Lemma 8 on $\widehat{P} \cup \mathring{\widehat{P}}^{\to i^0_r}$:

$$\begin{aligned} t_{|\widehat{P}|+|\mathring{\widehat{P}}^{\to i^0_r}|} &\geq e^{k_1} + \tau(\widehat{P}) + \tau(\mathring{\widehat{P}}^{\to i^0_r}) \\ &= e^{k_1} + \rho(P) + \rho(\mathring{P}^{\to \mathring{P}_{(r)}}). \end{aligned}$$

Following from the proof in Lemma 8, there exists $k' \in K$ such that

$$((n_r, t_{s^1-1}, t'_{s^1-1}), (n_1, t_{s^1}, t'_{s^1})) \in \mathcal{A}^{k'}_{\mathcal{T}'},$$

with $s^1 = |\widehat{P}| + |\mathring{\widehat{P}}^{\to i_1^1}|$. By $[\![5]\!]$ on $\mathcal{A}^{k'}_{\mathcal{T}'}$

$$\begin{aligned} t'_{s^1} &> t_{s^1-1} + \tau_{(n_r,n_1)} \\ &\geq e^{k_1} + \rho(P) + \rho(\mathring{P}^{\to \mathring{P}_{(r)}}) + \tau_{(n_r,n_1)} \\ &= e^{k_1} + \rho(P) + \rho(\mathring{P}). \end{aligned}$$

By Proposition 2, and since $(n_1,\, e^{k_1} + \rho(P) + \rho(\mathring{P})) \in \mathcal{T}'$ it follows that

$$t_s \geq e^{k_1} + \rho(P) + \rho(\mathring{P}).$$

Thus $\widehat{P} \cup \mathring{\widehat{P}}^{\to i_1^1}$ is a simple path in $(\mathcal{N}_{\mathcal{T}'}, \mathcal{A}^K_{\mathcal{T}'})$, intuitively then, for $m \leq M$ walks around the cycle, $\widehat{P} \cup \mathring{\widehat{P}}^{\to i_1^m}$ is still a simple path and thus

$$\begin{aligned} t_{s^m} &\geq e^{k_1} + \tau(\widehat{P}) + \tau(\mathring{\widehat{P}}^{\to i_1^m}) \\ &= e^{k_1} + \rho(P) + m\,\rho(\mathring{P}), \end{aligned}$$

with $s^m = |\widehat{P}| + |\mathring{\widehat{P}}^{\to i_1^m}|$, and $a = ((n_r, t_{s^m-1}, t'_{s^m-1}),\, (n_1, t_{s^m}, t'_{s^m})) \in \mathcal{A}^{k'}_{\mathcal{T}'}$ – however, by the choice of $\mathring{m}$, this means $t_{s^m} > l^{k_1}$ which violates $[\![4]\!]$ on $\mathcal{A}^{k_1}_{\mathcal{T}'}$. Thus $a \notin \mathcal{A}^{k_1}_{\mathcal{T}'}$, and so there does not exist cycle $\mathring{\widehat{P}}$ corresponding to $\mathring{P}$, hence by Observation 3, S is not representable in $\mathcal{G}^K_{\mathcal{T}'}$. □

4.6 Algorithms for Solving CTSNDP

The high level implementation details of DDDI are shown in Algorithm 6. To simplify the pseudocode, it is assumed that a feasible continuous-time solution exists (this can easily be checked by using shortest paths). The function BUILD_TIME-EXPANDED_MODEL uses the conditions defined in Section 4.3.2 to build the IP model defined by (4.1), which is then solved using an integer programming package in SOLVE_LOWERBOUND_MIP.

Algorithm 6 DDDI Algorithm

Require: Flat network $G = (N, A)$, Commodities K, Optimality tolerance DDDI_TOL. Pseudocode assumes that a feasible continuous-time solution exists

$\mathcal{T} \leftarrow N \times \{\min_{k \in K} e^k,\ \max_{k \in K} l^k + 1\}$
model $\leftarrow$ BUILD_TIME-EXPANDED_MODEL$(G, K, \mathcal{T})$
$(lower\_bound,\ upper\_bound) \leftarrow (-\infty,\ \infty)$

while True **do**
    flat-solution $\leftarrow$ SOLVE_LOWERBOUND_MIP(model)
    $lower\_bound \leftarrow \max\{lower\_bound,$ COST(flat-solution)$\}$

    if all paths in flat-solution are k-feasible **then**
        new-solution $\leftarrow$ CONVERT_SOLUTION(flat-solution)
        $upper\_bound \leftarrow \min\{upper\_bound,$ COST(new-solution)$\}$

    if $(upper\_bound - lower\_bound) < upper\_bound *$ DDD_TOL **then**
        Stop. The converted solution is within tolerance of the optimal

    $\mathcal{T} \leftarrow$ REFINE_DISCRETIZATION$(GS($flat-solution$), \mathcal{T})$
    model $\leftarrow$ UPDATE_MODEL(model, $\mathcal{T}$)

An *upper bound* flat-solution is continuous-time feasible (i.e. implementable), but not necessarily optimal. The non-implementable flat-solution $S = (Q^K, C)$ can easily be converted to an upper bound flat-solution when all of the paths in Q^K are k-feasible – since in this scenario, the non-implementability is entirely due to inappropriate consolidations. The CONVERT_SOLUTION(S) function creates a upper bound flat-solution, by finding a corresponding timing for S such that commodities having an inappropriate consolidation are dispatched independently (i.e. the consolidation is *broken* or *unpacked*). This corresponding timing is found by solving the following LP, which closely matches the implementable inequalities in (4.2).

$$\min \sum_{(a,\kappa)\in C} \sum_{k_1,k_2\in\kappa} f_a \pi_a^{k_1,k_2}, \tag{4.5a}$$

subject to

$$\begin{aligned}
t^k_{o^k} &\geq e^k, && \forall k \in K, && (4.5b)\\
t^k_{n_2} &\geq t^k_{n_1} + \tau_{(n_1,n_2)}, && \forall k \in K,\ (n_1,n_2) \in Q^k, && (4.5c)\\
t^k_{d^k} &\leq l^k, && \forall k \in K, && (4.5d)\\
t^k_n &\geq 0, && \forall k \in K, n \in Q^k, && (4.5e)
\end{aligned}$$

$$\begin{aligned}
\pi_a^{k_1,k_2} &\geq t^{k_1}_{n_1} - t^{k_2}_{n_1}, && \forall((n_1,n_2),\kappa) \in C,\ k_1,k_2 \in \kappa, && (4.5f)\\
\pi_a^{k_1,k_2} &\geq t^{k_2}_{n_1} - t^{k_1}_{n_1}, && \forall((n_1,n_2),\kappa) \in C,\ k_1,k_2 \in \kappa, && (4.5g)\\
\pi_a^{k_1,k_2} &\geq 0, && \forall(a,\kappa) \in C,\ k_1,k_2 \in \kappa, && (4.5h)
\end{aligned}$$

The objective (4.5a) minimizes the weighted cost of breaking a consolidation, that is, it attempts to preserve consolidations that incur a high cost to unpack. The constraints (4.5b), (4.5c), and (4.5d) ensure that each commodity leaves its origin node after e^k and reaches its destination node before l^k. Consolidations are softly enforced in (4.5f), and (4.5g) by measuring the difference between dispatch times. Lastly, (4.5e) and (4.5h) give non-negativity constraints.

If the upper bound flat-solution has a cost within the continuous-time optimal tolerance, then the algorithm terminates; otherwise the discretization needs to be refined. This most pivotal step is contained in the call to REFINE_DISCRETIZATION. Observe that for the non-implementable flat-solution $S = (Q^K, C)$, there may be many possible choices of paths, cycles, or arcs to apply Theorems 4, 5, 6, and 7. This choice has a huge impact on the solve time for an iteration, and the number of iterations required for convergence. Intuitively, adding fewer time points induces a smaller MIP model with faster solve times, but can also increase the number of iterations – potentially taking longer overall to converge to the continuous-time optimal solution. Thus it is important to develop a clever *refinement strategy* when implementing these theorems. The following subsections present various

strategies that focus on different issues: Default, Reduced Iterations, Fewer Time Points, and Adaptive.

Lastly, in our implementation, updates to the MIP model (UPDATE_MODEL) are performed in-place (not regenerated each iteration) and in a manner so that variables and constraints are reused as much as possible, that is, variables are typically relabeled instead of added/deleted. This reduces time spent generating the model, as well as potentially allowing the solver (Gurobi or CPLEX) to perform a warm start for the next iteration.

4.6.1 Default Refinement Strategy

The Default implementation is a direct translation of the main theorems. Here, the solution graph is first checked for cycles (Theorem 7), and if acyclic, it is checked for violated paths (Theorem 4), in either case adding time points as prescribed. Only one violation is fixed per iteration, chosen in order of the commodity index.

Algorithm 7 Default refinement strategy

1: **procedure** REFINE_DISCRETIZATION($GS(S), \mathcal{T}$)
2: **if** a cycle exists in $GS(S)$ **then**
3: **return** $\mathcal{T}'$ as per Theorem 7
4: **else if** a failed path exists in $GS(S)$ **then**
5: **return** $\mathcal{T}'$ for a single path as per Theorem 4
6: **return** $\mathcal{T}$

4.6.2 Reduced Iterations

One way to view the algorithm is that non-implementable flat-solutions are induced by *issues* in the time-expanded network, which are *fixed* by refining the discretization. Since there are many reasons why a flat-solution is non-implementable, it stands to reason that there are many corresponding issues in the time-expanded network that can be fixed. A natural extension to the Default algorithm fixes multiple issues in each iteration, which intuitively reduces the total number of iterations required for convergence.

If the solution graph is composed of disconnected subgraphs, then the commodities in different components do not interact, and so, adding time points to fix an issue in one component is unlikely to fix issues in other components. With this in mind, adding the set of fixes, one from each component (if applicable), should reduce the number of required iterations without unnecessarily increasing the associated time-expanded network.

Extending this idea further, it can also be beneficial to add several fixes from within a component, assuming the component is acyclic. We say that the consolidation (a_2, κ_2) is *downstream* from the consolidation (a_1, κ_1) if there exists a directed path in the solution graph connecting $s_{a_1}^{\kappa_1}$ to $s_{a_2}^{\kappa_2}$. Similarly, (a_1, κ_1) is said to be *upstream* from (a_2, κ_2). Furthermore, we say that the commodity k has *failed* if its solution path Q^k is not k-feasible; and say that the consolidation (a, κ) has *failed* if the last commodity to arrive at the consolidation is later than the earliest time another commodity must to leave (to reach its destination in time). We refer to these failed commodities and failed consolidations as *failures*.

A failure, and its corresponding fix, is considered *independent* if there exists no other failures upstream. Thus for an acyclic component, it is suitable to add all the independent fixes, without having too much concern for redundancy.

The last improvement focuses on the path used in Theorem 4. This path is chosen by walking up the solution graph from $t_{d^{k_2}}^{k_2}$, for some commodity, k_2, that arrives late in the given solution, and follows the $\arg\max_{k \in \kappa} E_{t_n^k}$ commodity at each consolidation. Observe that there may be multiple paths that match this definition, and while each of these paths are suitable, there may be one that is better than the others. A reasonable strategy is to simply use all paths. While this adds some redundancy, empirical evidence suggests that the benefits typically outweigh any overheads.

Algorithm 8 DDDI with Reduced Iterations

```
1: procedure REFINE_DISCRETIZATION(GS(S), T)
2:     T' ← T
3:
4:     for all CS ∈ CONNECTED_COMPONENT_SUBGRAPHS(GS(S)) do
5:         if a cycle exists in CS then
6:             Add time points to T' as per Theorem 7
7:         else
8:             for all independent failures in CS do
9:                 Add time points to T' as per Theorem 4 (all suitable paths)
10:    return T'
```

4.6.3 Fewer Time Points

In DDDI, the majority of the computational effort is spent solving the lower bound IP. Since this solve time is strongly correlated with the size of the model, it is preferable to keep the discretization as small as possible. By a careful application of Theorems 5 and 6, it becomes possible to fix more issues in each iteration using far fewer time points – and hence directly reducing the total, and per iteration, solve times. Since both these theorems use a single time point to fix a failure, we will refer to these as *unary* time points, and *unary* failures (respectfully).

By including these unary time points, the number of options for fixing failures increases dramatically. Building upon the Reduced Iterations strategy, we again prefer fixes that are upstream. Intuitively, any failures that occur earlier in the discretization (i.e. furtherest upstream) have more significance, due to the flow-on effects for paths and consolidations downstream. Since Theorems 5 and 6 do not assume an acyclic solution graph, our algorithm prioritizes fixing unary failures, preferring those as upstream as possible, before checking for cycles or adding time points from Theorem 4. Note that this preference for unary fixes means that unary failures that are downstream from non-unary failures can still be chosen.

For the failed consolidation (a, κ), observe that there are potentially $\binom{|\kappa|}{2}$ ways to apply Theorem 6. We attempt to chose the corresponding unary time point in a clever way. We

choose the pair of commodities with the smallest time-window gap. The *time-window* for commodity $k \in K$ at node $n \in Q^k$ is defined as:

$$[e_n^k,\, l_n^k] = \left[e^k + \tau\left((Q^k)^{\to n}\right),\quad l^k - \tau\left(Q^k\right) + \tau\left((Q^k)^{\to n}\right)\right]$$

that is, e_n^k is the earliest time to reach node n along path Q^k, and similarly l_n^k is the latest time to leave node n and still reach the destination in time. This interval is well defined when Q^k is a k-feasible path. Thus the time-window gap for a pair of commodities $k_1, k_2 \in \kappa$ on consolidation $((n_1, n_2), \kappa) \in C$ is given by

$$\left|e_{n_1}^{k_2} - l_{n_1}^{k_1}\right|.$$

Consider the example consolidation in Figure 4.13 where each commodity's time-window is shown. There are up to four ways to apply Theorem 6, using commodity pairs $(k_1, k_3), (k_1, k_4), (k_2, k_3)$, or (k_2, k_4). Our heuristic will choose (k_2, k_3), and hence partitions the consolidation evenly, preserving the clusters of overlapping commodities.

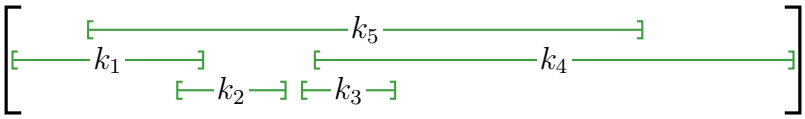


Figure 4.13: Disjoint Time-Windows for a Consolidation

The intuition behind Theorem 6 is that the consolidation $((n_1, n_2), \kappa)$ has disjoint time-windows, that is, $\cap_{k\in\kappa}[e_{n_1}^k, l_{n_1}^k] = \emptyset$. Moreover, it may be possible that κ has many clusters of commodities with non-overlapping time-windows, thus there are potentially many commodity pairs, $k_1, k_2 \in \kappa$, with disjoint time-windows, i.e. $e_{n_1}^{k_2} > l_{n_1}^{k_1}$. We choose the commodity pair that satisfies Theorem 6 with the smallest time-window gap, as this is likely to preserve many clusters either side of the partition.

Note that it may be possible to fix a cycle using unary time points from Theorem 6. This is preferred, since Theorem 7 requires potentially very many time points, due to the recursive nature of cycles. Although only a single unary time point may be sufficient to change a cycle, we have found that fixing every unary failure (around the cycle) is more effective at breaking it completely in one step, while still using far fewer time points than

Theorem 7.

For cycles without unary failures, the number of time points can still be reduced by simply checking at each node whether any commodity (i.e. not only the one mentioned in Theorem 7) participating in the cycle, has a shortest path to its destination that exceeds its late time; when this is found, no further time points are required. A similar idea can be applied to Theorem 4, that is, by using shortest paths it is possible to skip a number of nodes at the start and end of a violated path.

Algorithm 9 DDDI with Fewer Time Points

procedure REFINE_DISCRETIZATION($GS(S), \mathcal{T}$)
    $\mathcal{T}' \leftarrow \mathcal{T}$

    for all $k \in K$ **do**
        Check Theorem 5 along path, add first time point (if exists) to $\mathcal{T}'$

    for all unary failed consolidations **do**
        if consolidation is not downstream from another unary failure **then**
            Check Theorem 6 (add to $\mathcal{T}'$ using prescribed heuristic)

    for all $CS \in$ CONNECTED_COMPONENT_SUBGRAPHS($GS(S)$) **do**
        if cycle exists in CS that is not downstream a failure **then**
            if cycle can be broken using unary time points **then**
                Add time points to $\mathcal{T}'$ as per Theorem 6
            else
                Add time points to $\mathcal{T}'$ as per reduced Theorem 7
        else
            for all independent failures in CS not downstream from any fixes **do**
                Add time points to $\mathcal{T}'$ as per reduced Theorem 4 (all suitable paths)
    return $\mathcal{T}'$

4.6.4 Reduced Time-Expanded Network

The fixed and variable costs in our definition of CTSNDP are constant over time, thus the total cost of an implementable solution does not depend on a corresponding timing. It suffices then, to only consider dispatch times that occur as early as possible, meaning that some arcs in the time-expanded network are redundant.

In the algorithm below, $D^{i,n}$ is defined, for $i \in \mathcal{N}_{\mathcal{T}}$ and $n \in N$, as the set of timed-arcs having origin node-interval i, and a destination node-interval with node n; $D^{i,n}_{(j)}$ is defined as the jth arc in $D^{i,n}$ ordered by time; and $K^{i,n}_{(j)}$ is the set of commodities that have the ability to travel on arc $D^{i,n}_{(j)}$.

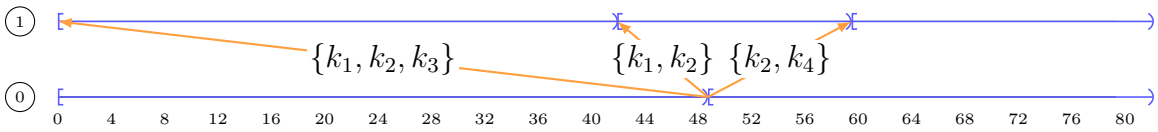


Figure 4.14: Redundant Consolidation Arc

The idea is that if $K^{i,n}_{(j)}$ contains all commodities in $K^{i,n}_{(j+1)}$, i.e. $K^{i,n}_{(j)} \supseteq K^{i,n}_{(j+1)}$, then $D^{i,n}_{(j+1)}$ is redundant, since $D^{i,n}_{(j)}$ can dispatch all the commodities earlier. Figure 4.14 shows an example where the $\{k_1, k_2\}$ arc is redundant due to the $\{k_1, k_2, k_3\}$ arc dispatching earlier, but $\{k_2, k_4\}$ is still required because of the addition of k_4.

Algorithm 10 Redundant arcs

```
1: procedure REMOVE_REDUNDANT_ARCS
2:     for all i = (n_1, t_1, t'_1) ∈ N_T do
3:         for all n ∈ N \ {n_1} do
4:             for all j ∈ {1, ..., |D^{i,n}| − 1} do
5:                 if K^{i,n}_{(j)} ⊇ K^{i,n}_{(j+1)} then
6:                     mark D^{i,n}_{(j+1)} as redundant
7:     delete redundant arcs
```

It is the hope that by removing these redundant arcs, the time-expanded model can be significantly reduced and the MIP solve times improved.

4.6.5 Adaptive Refinement Strategy

Our final, and most complete, refinement strategy uses the approach from Fewer Time Points (Algorithm 9) on the reduced time-expanded network (Section 4.6.4), as well as incorporating an adaptive tolerance when solving the lower bound IP.

To avoid confusion, we define the following terms: MIP_GAP, MIP_TOL, DDDI_GAP, and DDDI_TOL. Every iteration of the DDDI algorithm solves an IP with a target MIP_TOL; we refer to the IPs optimality gap (after terminating the solve) as the MIP_GAP. Each iteration in Algorithm 6 has DDDI_GAP $= \frac{upper\_bound - lower\_bound}{upper\_bound}$, which is the continuous-time optimality gap. The algorithm terminates when DDDI_GAP $<$ DDDI_TOL.

Typically when solving instances, the first several iterations have a large DDDI_GAP, and so there is very little benefit in using a small MIP_TOL. Instead of having a fixed MIP_TOL across all iterations, it makes sense to use the DDDI_GAP as a guide. Our implementation starts with MIP_TOL $= 0.04$, and then for each iteration:

$$\text{MIP\_TOL} = \max\{\text{DDDI\_GAP} * 0.25,\ \text{DDDI\_TOL} * 0.98\}.$$

These values have been chosen based on empirical experimentation. In particular notice that MIP_TOL can be set to a value strictly less than DDDI_TOL. This is not required but has been found (empirically) to improve convergence in some cases.

4.7 Computational Study

We solve the 558 unrounded (i.e. 1 minute) CTSNDP instances from Section 2.4, to 0.001% optimality gap, with a computational time limit of 1 hour. Memory is limited to the available system memory (256 Gb), and processing is restricted to a single core. CPLEX 12.6 was used as the MIP solver.

Each instance was solved by the full discretization, as well as all the various refinement strategies mentioned above (Default, Reduced Iterations, Fewer Time Points, and Adaptive), and the results were grouped by flexibility and cost ratio. Chapter 2 showed that these metrics characterize discretization gap and tractability reasonably well, and so grouping by this scheme can highlight performance characteristics that might otherwise be lost by taking the average across all instances. In what follows, an instance is said to have low flexibility (LF) if

$$\min_{k \in K} l^k - e^k - \gamma_1(o^k, d^k) < 227,$$

and low cost ratio (LC) if

$$\frac{1}{|A|}\sum_{a\in A}\frac{f_a}{v_a u_a} < 0.175.$$

The choice in parameters has been determined by some of the results in Chapter 2, as well as trying to equally spread instances across the groups. The resulting group sizes can be seen in Table 4.6.

Table 4.6: Group allocation of the 558 instances

	LF	**HF**
HC	183	177
LC	94	104

The summary of results, averaged for each of the groups, can be found in Table 4.7. Note that the time required to build the model was not included in the computational time limit; some instances required over 30 minutes to build the full discretization model, whereas DDDI barely takes seconds. Also note that due to the computational time limit, the full discretization IP (after building the model), did not produce a feasible solution for 28 instances (all in the HC/HF group). For these instances a gap of 100% was used for the results; this choice seemed reasonable, since many similar instances from this group had a gap of over 100%, and so the results should be not skewed in favor of DDDI.

As can be seen by the results, DDDI clearly outperforms solving the full discretized model. Also, there are significant differences between the refinement strategies; in terms of number of iterations and the average gap %. As expected, instances with high cost and high flexibility are the most difficult to solve, however DDDI still does a remarkable job finding high quality solutions for these instances within the time limit.

Table 4.7: Computational Results DDDI Variants vs Full Discretization (1hr solve time)

Algorithm	Gap %	Time (s)	# Iterations	% Optimal
HC/LF				
Full Discretization	18.28	2,595.03		32.8
Default	3.53	1,751.68	92.6	60.1
Reduced Iterations	1.70	1,386.03	11.2	70.5
Fewer Time Points	1.34	1,258.35	11.2	74.3
Adaptive	0.12	677.76	14.8	85.8
HC/HF				
Full Discretization	49.69	3,290.23		12.4
Default	13.12	2,224.00	53.6	45.2
Reduced Iterations	8.94	2,021.67	10.5	51.4
Fewer Time Points	8.53	1,991.27	11.9	48.6
Adaptive	0.84	1,693.76	17.5	56.5
LC/LF				
Full Discretization	0.00	270.24		95.7
Default	0.00	3.85	28.0	100.0
Reduced Iterations	0.00	1.01	6.0	100.0
Fewer Time Points	0.00	0.67	6.1	100.0
Adaptive	0.00	0.59	6.5	100.0
LC/HF				
Full Discretization	0.00	715.94		93.3
Default	0.00	0.39	7.8	100.0
Reduced Iterations	0.00	0.18	3.4	100.0
Fewer Time Points	0.00	0.12	3.0	100.0
Adaptive	0.00	0.13	3.2	100.0

It is interesting to notice that for HC/HF, the Fewer Time Points strategy solves less instances to optimality than Reduced Iterations, however the average gap of Fewer Time Points is less. This highlights the delicate balance between convergence and model size, and the difficulty in choosing a suitable refinement strategy. That is, in some instances the extra time points (of Reduced Iterations) helped prove optimality in less time, however on average the smaller model size of Fewer Time Points produced solutions with tighter optimality gaps.

To further compare the various refinement strategies, Figures 4.15 and 4.16 show the convergence and model growth of the approaches, averaged over time.

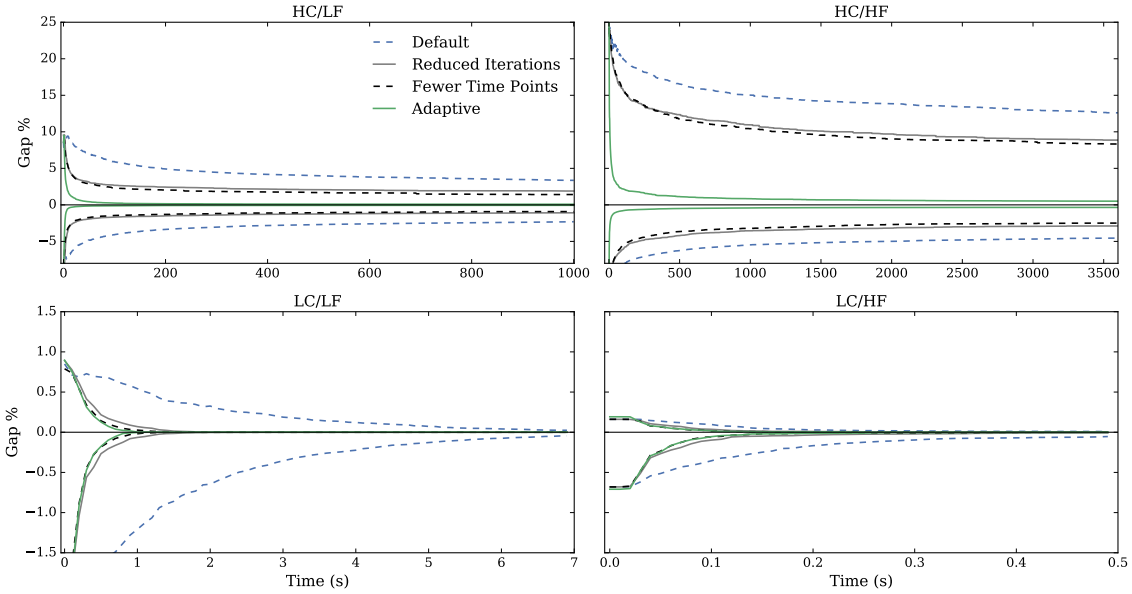


Figure 4.15: DDDI Convergence (1hr solve time)

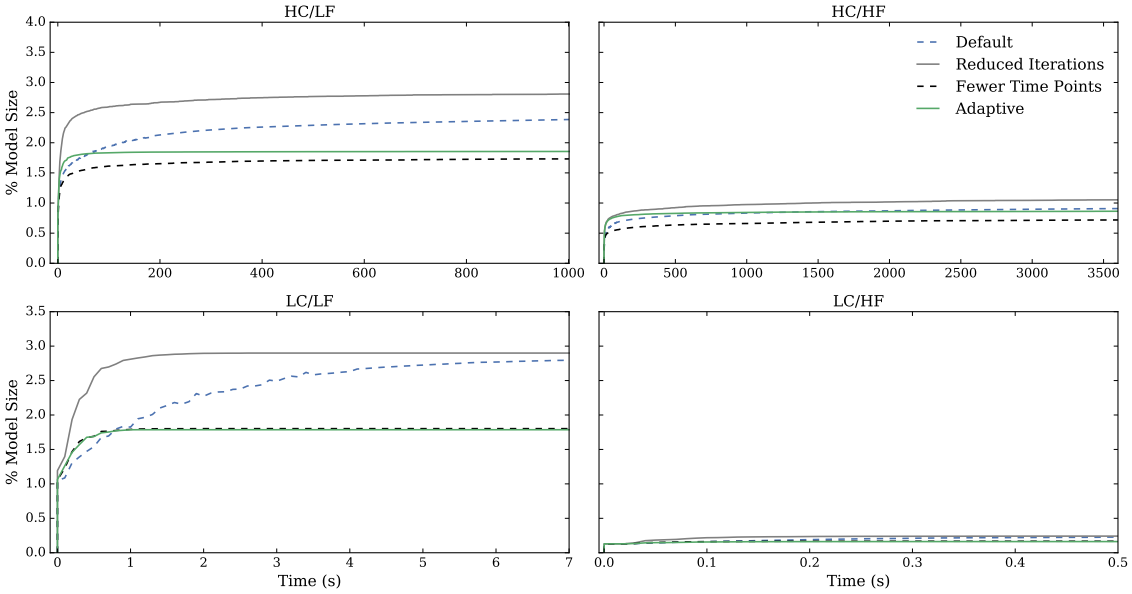


Figure 4.16: DDDI Model Growth (1hr solve time)

Note that the Adaptive strategy has a remarkably quick initial convergence, even with the HC/HF instances. Much of the time is spent proving optimality. Since our algorithm

typically produces feasible solutions at every iteration, it is obvious that DDDI also makes for an excellent heuristic – finding good solutions quickly.

The model growth is measured by the total number of variables and constraints, relative to the full discretization model. Note that the full discretization model has also been constructed using the conditions in Section 4.3.2, and so many redundant arcs have also been eliminated. That is, if the full discretization model was constructed naively, the relative size comparison would be even more significant.

As expected, Figure 4.16 shows that on average the Reduced Iterations and Fewer Time Points induce the largest and smallest (resp.) models sizes. In all cases, most growth occurs near the start of the algorithm. Note that the average full discretization model size for each group is different, and so the groups cannot be directly compared to each other, however even if these results are relatively weighted, we notice a similar pattern.

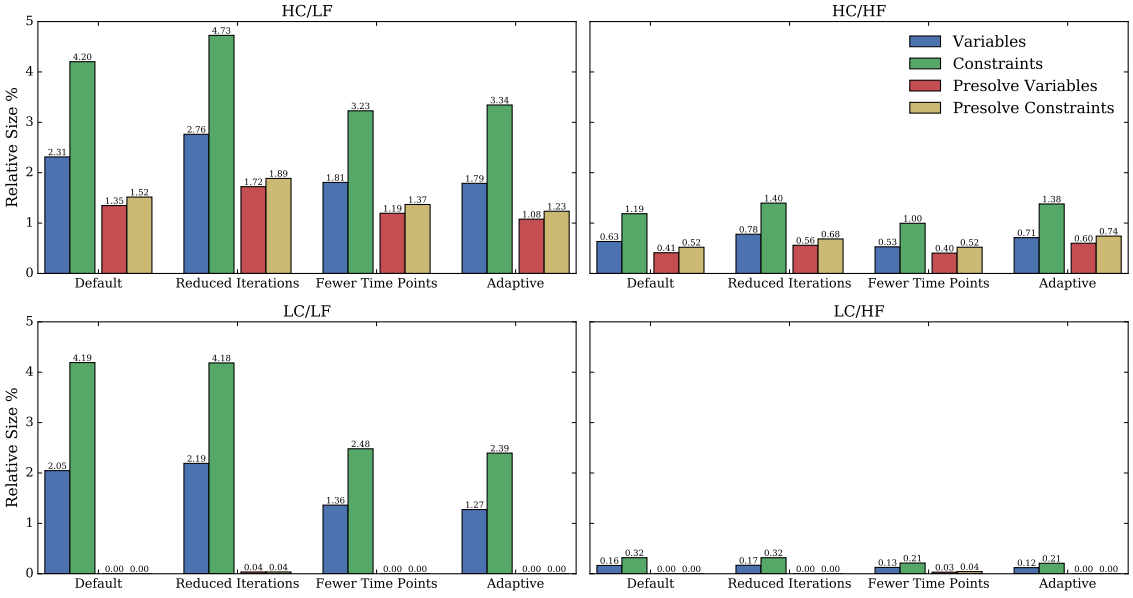


Figure 4.17: DDDI Model Size (1hr solve time)

Lastly, Figure 4.17 shows the model size (relative to the full discretization) of the IP from the last iteration, averaged over each strategy. The Presolve Variables/Constraints gives the relative size of the model after CPLEX has performed its presolve routines

(identifying and removing redundant variables/constraints). Note that these are relative to the presolved model of the full discretization, so that it is a fair comparison. Thus in all cases, DDDI produces models that are significantly smaller, typically over 100 times smaller, which makes it a suitable approach for solving large scale instances. Observe that the presolve variables and constraints in the LC/LF and LC/HF groups, are practically at 0%. In many instances, CPLEX could remove all the variables and constraints for the final DDDI model, whereas the presolved full discretization model often remained large. There were a number of instances where the full discretization had also presolved away all variables and constraints; these instances were removed (for those columns only), to avoid a divide by zero. This shows that DDDI does exceptionally well for instances with low cost ratios.

For the implementation of the above algorithms, we have included an additional valid inequality in the model that is used tighten the formulation:

$$\left\lceil \frac{q^k}{u_a} \right\rceil x_a^k \leq z_a, \qquad \forall k \in K, a \in \mathcal{D}_{\mathcal{T}}^k \tag{4.6}$$

To see the effect of these extra constraints, we performed the above calculations without the valid inequality, using the Adaptive refinement strategy. The comparative results can be seen in Table 4.8. Clearly it has a significant effect on convergence. When using the cuts the average gap % is nearly half, and around 10% additional HC instances can be solved to optimality – however the number of constraints (relative to the full discretization) is doubled, albeit it is still very small. For extremely large instances, these cuts can be removed in order to reduce the model size, however these results highlight the benefits of these extra constraints.

Table 4.8: Results with/without cuts (1hr solve time)

Algorithm	Gap %	Time (s)	Constraints %	# Iterations	% Optimal
HC/LF					
Adaptive (w/o cuts)	0.21	1,209.76	1.35	14.1	73.2
Adaptive	0.12	677.76	3.34	14.8	85.8
HC/HF					
Adaptive (w/o cuts)	1.61	2,005.20	0.50	15.7	46.9
Adaptive	0.84	1,693.76	1.38	17.5	56.5
LC/LF					
Adaptive (w/o cuts)	0.00	0.84	1.15	6.8	100.0
Adaptive	0.00	0.59	2.39	6.5	100.0
LC/HF					
Adaptive (w/o cuts)	0.00	0.14	0.11	3.3	100.0
Adaptive	0.00	0.13	0.21	3.2	100.0

4.8 Comparison with DDD

Clearly the approaches in Section 3 (DDD) and Section 4 (DDDI) share many similarities. Both algorithms iteratively build and refine a time-expanded network, and solve the associated integer program to obtain a lower bound. Both relax transit time in order to guarantee the lower bound, however they enforce this in quite different ways. For the following comparison, unless otherwise specified, we adopt the notation from DDDI.

We argue that DDDI does not directly support any of the four properties required by DDD, and as a result, the lower bound IP models are not equivalent. Moreover, as will be discussed, the discretization refinement process is also different between the two approaches, even though they both make use of Proposition 4 (DDD) for the choice of time points.

Recall that Property 1 (DDD) requires time points (o^k, e^k), (d^k, l^k), and $(n, 0)$ for all $n \in N$ to exist, whereas DDDI only requires the existence of $(n, 0, T)$ for $n \in N$. DDDI implicitly incorporates the effect of having time points (o^k, e^k), (d^k, l^k) via the timed-arc conditions $[\![3]\!]$ and $[\![4]\!]$. As a consequence, and coupled with $[\![5]\!]$, it is possible for DDDI to

have a timed-arc $((n_1, t_1, t'_1), (n_2, t_2, t'_2))$ such that $t_2 > t_1 + \tau_{n_1,n_2}$, that is, transit time in the network is longer than the actual transit time – which violates Property 2 (DDD). Note that if these arcs exist (they are typically discarded by Algorithm 10), then there is often another timed-arc in DDDI that satisfies Property 2 (DDD), i.e. $((n_1, t_1, t'_1), (n_2, t_3, t'_3))$ such that $t_3 \leq t_1 + \tau_{n_1,n_2}$, and hence DDDI violates Property 4 (DDD) by allowing multiple arcs from the same node-interval to the target node n_2. Even if DDDI (using Algorithm 10) includes all the time points of Property 1 (DDD), then Property 2 (DDD) still does not always hold. Finally, Property 3 (DDD) is partially supported by DDDI via $[\![1]\!]$ and $[\![2]\!]$, in the sense that DDDI only requires a subset of the arcs in Property 3 (DDD).

DDDI has been designed to keep the model as small as possible, and the initial time-expanded network for each commodity is typically smaller than the original *flat* network. The conditions for timed-arcs have been chosen to reduce the size of the time-expanded network, and, where possible, DDDI attempts to exploit these conditions to effectively refine the discretization, while minimizing the growth of the network. The associated IP is also reduced (compared to DDD), since the path length constraints required by DDD, in Equations (3.5), are not needed in DDDI. Note that empirical evidence suggests that adding these constraints increases the time to solve the IP. Without these constraints, the DDD arc-lengthening IP model (used to refine the discretization) is not guaranteed to be feasible. One advantage of this arc-lengthening IP model (and the DDD approach), is that it allows for easy implementation, with much of the algorithmic burden able to be performed by powerful commercial solvers. While this approach is effective and easy to extend, it does not fully exploit the structure of lower bound flat-solutions. Proof of convergence for DDD relies on finite number of arc-length extensions, whereas DDDI uses the finite number of flat-solutions. As such, it is possible for DDD to return the same flat-solution for many iterations (unlike DDDI), and so it is clear that the time points chosen by DDD are not equivalent to those chosen by DDDI. Moreover, DDDI converges (albeit slower, with many extra iterations) if using the arc-lengthening approach from DDD (and the path

length routines from DDDI), but the converse is not always true. To lengthen the timed-arc $((n_1, t_1, t'_1), (n_2, t_2, t'_2))$ in the DDDI setting, two time points are required: $(n_1, t_1 + 1)$, and $(n_2, t_1 + \tau_{n_1,n_2})$.

In an effort to reduce the number of iterations, the implementation of DDD makes effective use of multiple optimal solutions (when available) to the arc-lengthening IP, however this still does not prevent the same lower bound flat-solution from returning over multiple iterations.

When adding extra time points to refine the discretization, there is a precarious balance between the performance benefits of reducing the number of iterations, and the performance overhead of larger instance sizes. Both algorithms try to incorporate refinement strategies to achieve this balance, however it is difficult to find the best approach for all instances. DDDI has much greater flexibility in designing this strategy, but the additional power also comes with increased difficulty in choosing the best approach.

Lastly, both algorithms use the same upper bound LP, except that DDDI also incorporates a weight in the objective for breaking consolidations. By doing so, it attempts to preserve consolidations that incur a high cost to unpack, in the hope of finding a tighter upper bound.

To summarize, the two approaches, DDD and DDDI, are not equivalent. DDD is an effective and easy to implement algorithm, making good use of general purpose integer programming solvers. The implementation of DDDI is more involved, however the associated model sizes are considerably smaller – thus larger instances can be supported, with potentially quicker convergence.

4.8.1 Empirical Results

Each of the previously mentioned instances were solved by DDD, using the same computational limits, and compared with the results from DDDI (Adaptive variant), again grouping by flexibility and cost ratio.

The summary of results, averaged for each of the groups, can be found in Table 4.9. The model size information (variables, constraints) is obtained for each instance from the IP of the last iteration. As expected, the HC instances are more difficult to solve, where the HC/HF is the most difficult. DDDI solves more instances to optimality, with a much smaller model size. Notice that DDD has more than 12 times the number of variables and over 5 times the number of constraints. That said, on average DDD has a lower gap %; meaning that for the instances that did not solve to optimality, DDD found tighter solutions. Specifically, when comparing to the continuous-time optimal value, on average DDDI was found to have tighter lower bound values, while DDD had a tighter upper bound.

Table 4.9: Computational Results DDD vs DDDI (1hr solve time)

Algorithm	**Gap %**	**Time (s)**	**Variables**	**Constraints**	**# Iterations**	**% Optimal**
HC/LF						
DDD	0.08	1,391.1	205,540.2	177,149.5	5.3	77.1
DDDI	0.12	677.8	14,013.6	25,757.0	14.8	85.8
HC/HF						
DDD	0.56	1,966.7	161,097.2	128,888.2	6.0	53.7
DDDI	0.84	1,693.8	13,044.6	23,496.2	17.5	56.5
LC/LF						
DDD	0.00	28.6	20,633.5	27,177.9	3.7	100.0
DDDI	0.00	0.6	1,382.4	2,535.4	6.5	100.0
LC/HF						
DDD	0.00	1.5	4,500.3	6,917.1	2.5	100.0
DDDI	0.00	0.1	376.7	639.0	3.2	100.0

The difference in gap % is further explored in Figure 4.18, by looking at the average convergence profiles over time. Notice how DDDI converges rapidly, but in the HC/HF instances it struggles to improve the solution after a certain point, where DDD (on average) continues to find better solutions. This may be an artifact of the 1 hour computational limit, or it might be a side effect of the extra variables and constraints employed by DDD.

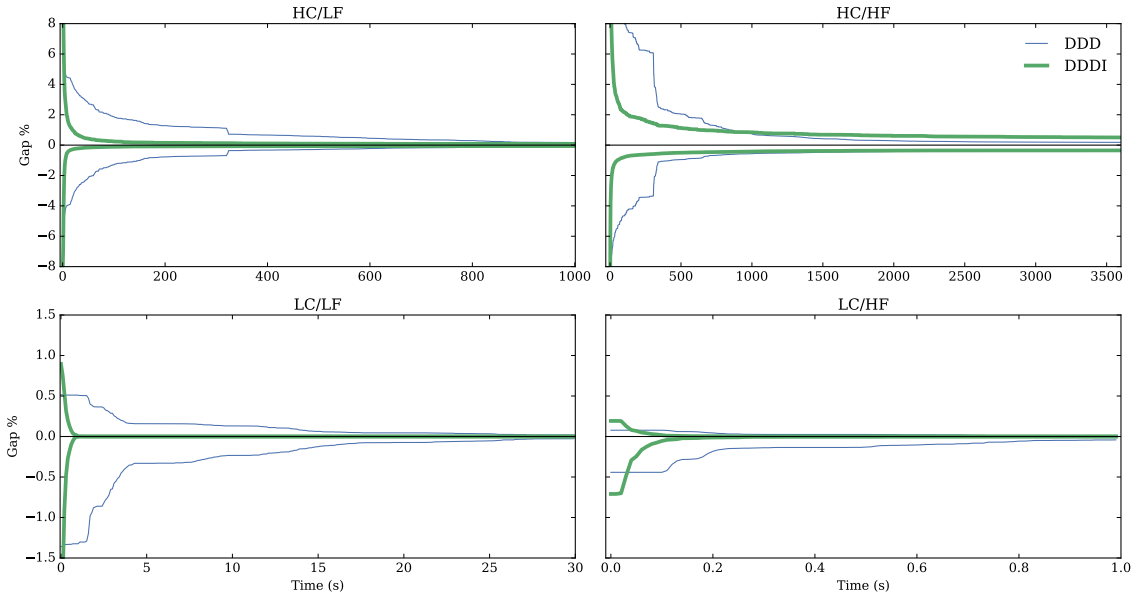


Figure 4.18: DDD vs DDDI Convergence (1hr solve time)

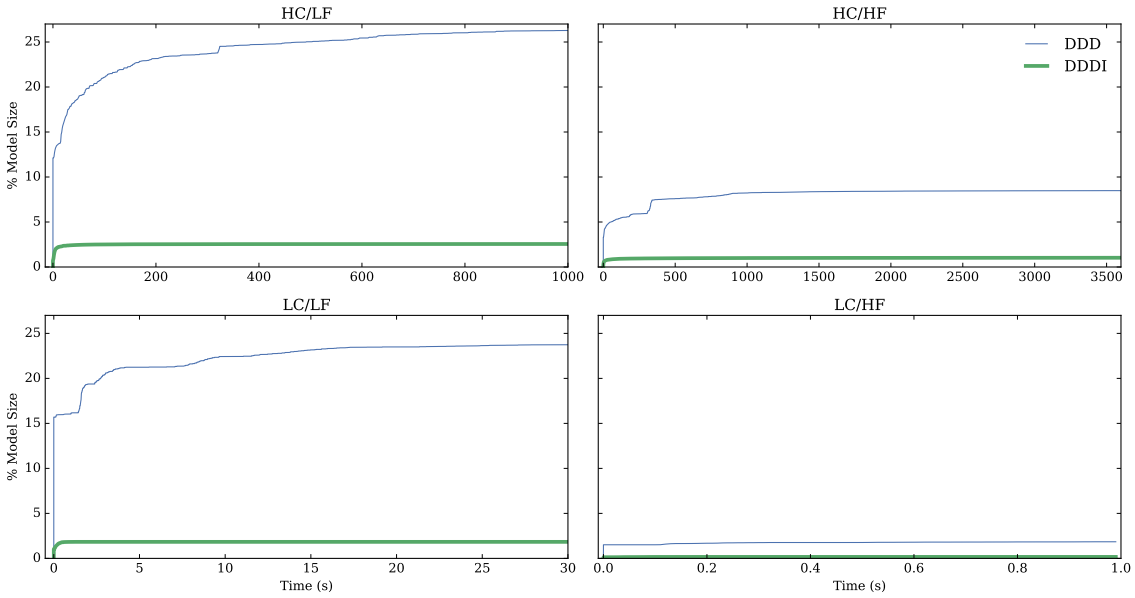


Figure 4.19: DDD vs DDDI Model Growth (1hr solve time)

In Figure 4.19 the growth of the model size (variables + constraints) is compared over time, as a percentage of the full discretization model. Note that the full model size for each group is different, and so the groups cannot be directly compared to each other, however even if these results are relatively weighted, we notice a similar pattern. On average, the

HF instances require a smaller percentage of the full discretization. While this may be partially due to the effect of different time horizons in the groups; this result is expected since our algorithms have been designed to efficiently handle large intervals of time in the model, which are intuitively more common in instances with high flexibility. Notice again that DDDI has a significantly smaller model size (than DDD), and seems to stabilize relatively quickly. Finally, in Figure 4.20, each of the 558 instances are plotted against the difference in both gap %, and time to solve (limited to 1 hour). Negative values on the left are instances where DDDI outperformed DDD. For example, a time of -2000 is interpreted as DDDI solving 2000 seconds quicker than DDD. Positive values on the right indicate instances where DDD outperforms DDDI. As already mentioned, for instances that are not solved to optimality, DDD tends to find better solutions within the hour (up to 3% difference); however, DDDI tends to solve more instances in much less computational time.

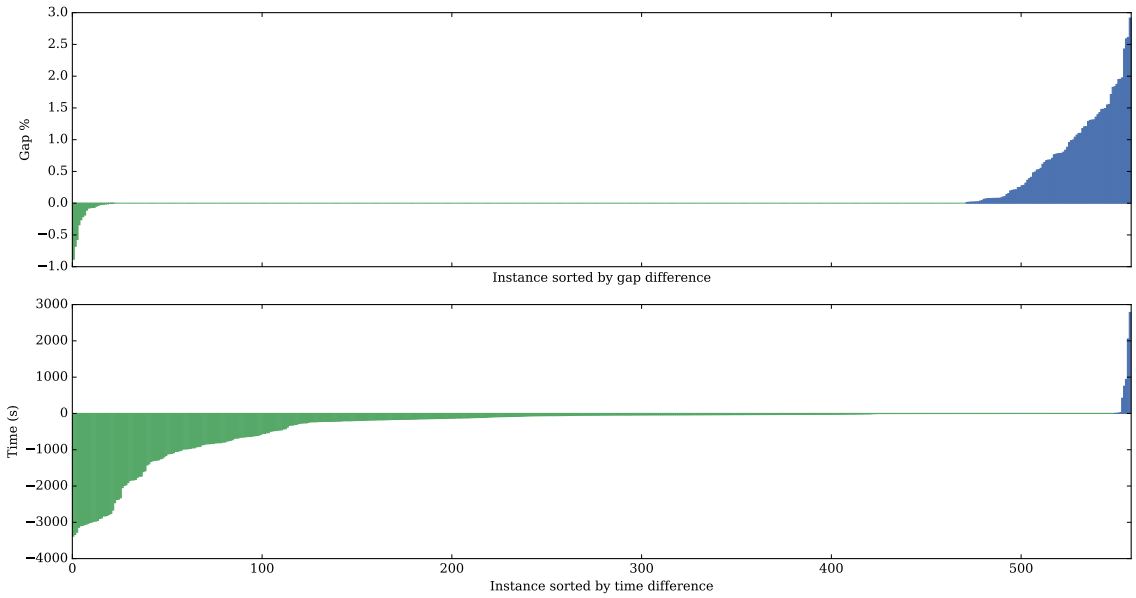


Figure 4.20: DDD vs DDDI Gap and Solve Time (1hr solve time)

Even though DDDI performs quite well, these results suggest that further exploration and extensions to the DDDI refinement strategy seem warranted and might have significant payoff. By adding more constraints, or extra time points at each iteration, it might be

possible to find optimal solutions faster, while still using model sizes smaller than those used by DDD. Furthermore, a different refinement strategy could be designed to adapt for very large instance sizes, and use less constraints/time points in order to solve the corresponding large scale integer programs.

CHAPTER 5

EXTENDING THE CONTINUOUS-TIME SERVICE NETWORK PROBLEM

This chapter presents two extensions that are applicable to both the DDD and DDDI algorithms (however the emphasis and implementation is on DDDI). This shows how the dynamic discretization technique can be applied to problems in a slightly different setting.

5.1 Commodity Splitting

In the discussion so far, commodities must be transported through the network as a single unit, however in practice it can be beneficial to allow commodities to be split into arbitrarily sized pieces that flow separately throughout the network. The ability to split commodities can result in reduced transport costs, by increasing trailer utilization, and it is often a modeling technique to handle larger volumes of freight (by aggregating shipments with common origin/destination), while keeping the model size tractable.

5.1.1 Model

To support splitting in the DDDI algorithm, the x variables in the lower bound IP model first need to be changed from binary to continuous, as seen in equations (5.1d) below.

$$\min \sum_{k \in K} \sum_{a \in \mathcal{A}^k_{\mathcal{T}}} v_a q^k x^k_a + \sum_{a \in \mathcal{D}^K_{\mathcal{T}}} f_a z_a, \tag{5.1a}$$

subject to

$$\sum_{a \in \delta^{k+}_i} x^k_a - \sum_{a \in \delta^{k-}_i} x^k_a = r^k_i, \qquad \forall k \in K, i \in \mathcal{N}_{\mathcal{T}}, \tag{5.1b}$$

$$\sum_{k \in K \,:\, a \in \mathcal{A}^k_{\mathcal{T}}} q^k x^k_a \leq z_a u_a, \qquad \forall a \in \mathcal{D}^K_{\mathcal{T}}, \tag{5.1c}$$

$$0 \leq x^k_a \leq 1, \qquad \forall k \in K, a \in \mathcal{A}^k_{\mathcal{T}}, \tag{5.1d}$$

$$z_a \in \mathbb{Z}_+, \qquad \forall a \in \mathcal{D}^K_{\mathcal{T}}. \tag{5.1e}$$

Solutions to the above can result in commodities that are split into multiple timed-paths. More specifically, for each commodity k, the solution to Equations (5.1) determines which timed-arcs have positive flow, that is $x_a^k > 0$. This set of timed-arcs can be taken as a subgraph, and then, using a simple depth first search, it is possible to enumerate all timed-paths. We wish to decompose this subgraph in order to create new temporary commodities corresponding to each timed-path, such that each new commodity has a quantity that appropriately matches the flow on the subgraph. Let $\mathcal{W}_{\mathcal{T}}^k$ be this set of timed-paths for commodity $k \in K$, that is, commodity k is split into $|\mathcal{W}_{\mathcal{T}}^k|$ commodities.

These new commodities exist for the purpose of refining the discretization and providing an upper bound for the current iteration only. By treating a commodity that has been split into multiple timed-paths as multiple independent commodities, the Theorems in Section 4.5 and Equations (4.5) hold without changes. The quantity of these temporary commodities cannot be determined directly from the solution to Equations (5.1), since there may be multiple timed-paths that share the same timed-arc. An appropriate allocation, for each $k \in K$, can be determined from the solution to Equations (5.2).

$$\sum_{P \in \mathcal{W}_{\mathcal{T}}^k} y_P^k = 1, \tag{5.2a}$$

$$\sum_{P \in \mathcal{W}_{\mathcal{T}}^k : a \in P} y_P^k = x_a^k, \qquad \forall\, a \in \mathcal{D}_{\mathcal{T}}^k, \tag{5.2b}$$

$$0 \leq y_P^k \leq 1, \qquad \forall\, P \in \mathcal{W}_{\mathcal{T}}^k, \tag{5.2c}$$

In the above, x_a^k is an input parameter, taken from the solution of Equations (5.1), and can be interpreted as the fraction of commodity k that flows on dispatch arc $a \in \mathcal{D}_{\mathcal{T}}^k$. Then, from the solution of Equations (5.2), the commodity associated with each timed-path $P \in \mathcal{W}_{\mathcal{T}}^k$, has the assigned quantity $y_P^k \cdot q^k$. These commodities are then used for the refinement and upper bound steps in the DDDI algorithm, after which they are discarded.

Note that when there is no variable cost (i.e. $v_a = 0$), it is possible that the solution to

Equations (5.1) contains cycles. These cycles are superfluous and should be removed when constructing $\mathcal{W}_T^k$, and the corresponding quantity in x_a^k should be updated (for each a in the cycle) to reflect this change. Without this, the Equations 5.2 may be infeasible.

5.1.2 Example

Consider the following example. The network in Figure 5.1 shows the transit times between nodes, the fixed cost is equal to transit time (except for $0 \rightarrow 1$ where the cost is 15); variable cost is zero, and capacity is one for all arcs. Commodities are shown in Figure 5.2, with quantity and time window $[e^k, l^k]$.

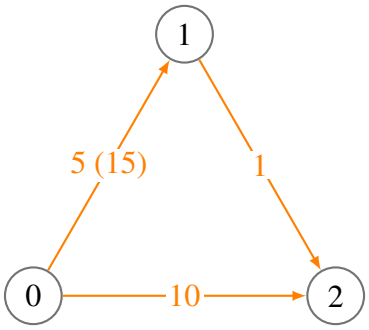


Figure 5.1: Network

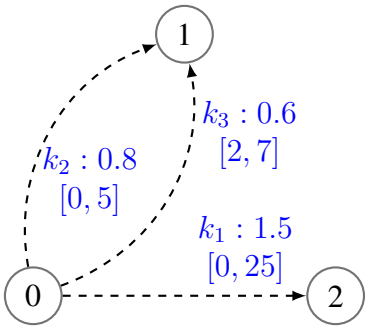


Figure 5.2: Commodities

The optimal solution can be seen in Table 5.3, where commodity k_1 is split into three pieces (quantities given by Table 5.4). Two pieces (k_1^2, k_1^3) travel along the same path, but having different dispatch times; whereas the third piece (k_1^1) travels along a different path. In this example, the optimal solution cost is 41, compared to 50 if splitting was not allowed.

Table 5.3: Split Example Solution

Arc	**Time**	**Consolidation** (k, q)	
0,2	0	$k_1, 1.0$	
0,1	0	$k_1, 0.1$	$k_2, 0.8$
0,1	2	$k_1, 0.4$	$k_3, 0.6$
1,2	7	$k_1, 0.5$	

Table 5.4: Temporary Commodities

k	k'	**Path**	$q^{k'}$
k_1	k_1^1	0,2	1.0
k_1	k_1^2	0,1,2	0.1
k_1	k_1^3	0,1,2	0.4

Note that the Inequality (4.6) is no longer valid when allowing freight splitting. Consider that k_1^1 requires only one dispatch along the arc $(0, 2)$, however Inequality (4.6) would require two dispatches since $x^{k_1}_{(0,2)} = \frac{1.0}{1.5}$ implies $z_{(0,2)} \geq \left\lceil \frac{1.5}{1.0} \right\rceil \frac{1.0}{1.5} = 1\frac{1}{3}$.

5.2 In-tree Loading

Optimal solutions to the CTSNDP may yield complex routing strategies for freight – especially if freight splitting is allowed. In order to implement these strategies in practice, advanced technological infrastructure is required to avoid time consuming and error prone physical handling at the terminals. Alternatively, a common approach used to avoid this issue is to instead reduce the number of potential choices for routing at a terminal. One such approach is in-tree loading. This enforces that the paths from all origins into one destination form a tree. In this way, the next terminal (for a commodity) can be determined simply by looking at its ultimate destination.

Consider the model below, for handling in-tree loading. For notation let $K(d)$ be the set of commodities that have destination d. That is, $K(d) = \{k \in K : d^k = d\}$.

5.2.1 Model

The model is very similar to the one in Section 4.2. Note that this formulation still works when commodities are allowed to be split. When splitting is not allowed, Inequality (4.6) holds as a valid inequality.

Auxiliary Variables

$$y_{d,n,n'} \begin{cases} 1 & \text{if the primary path for commodities with destination } d \in N \text{ contains } n \text{ to } n' \\ 0 & \text{otherwise} \end{cases}$$

Model

$$\min \sum_{k \in K} \sum_{a \in \mathcal{A}_{\mathcal{T}}^k} v_a q^k x_a^k + \sum_{a \in \mathcal{D}_{\mathcal{T}}^K} f_a z_a, \tag{5.3a}$$

s.t

$$\sum_{a \in \delta_i^{k+}} x_a^k - \sum_{a \in \delta_i^{k-}} x_a^k = r_i^k, \qquad \forall k \in K, i \in \mathcal{N}_{\mathcal{T}}, \tag{5.3b}$$

$$\sum_{k \in K \,:\, a \in \mathcal{A}_{\mathcal{T}}^k} q^k x_a^k \leq z_a u_a, \qquad \forall a \in \mathcal{D}_{\mathcal{T}}^K, \tag{5.3c}$$

$$\sum_{n' \in N} y_{d,n,n'} = 1, \qquad \forall\, n, d \in N, \tag{5.3d}$$

$$\sum_{a \in \mathcal{A}_{\mathcal{T}}^k \,:\, a=((n,\cdot,\cdot),(n',\cdot,\cdot))} x_a^k \leq y_{d,n,n'}, \qquad \forall\, n, n', d \in N,\, k \in K(d), \tag{5.3e}$$

$$y_{d,n,n'} \in \{0,1\}, \qquad \forall\, n, n', d \in N,\, k \in K(d), \tag{5.3f}$$

$$x_a^k \in \{0,1\}, \qquad \forall k \in K, a \in \mathcal{A}_{\mathcal{T}}^k, \tag{5.3g}$$

$$z_a \in \mathbb{Z}_+, \qquad \forall a \in \mathcal{D}_{\mathcal{T}}^K. \tag{5.3h}$$

Equation $(5.3d)$ chooses the next visited terminal $n' \in N$ at each terminal $n \in N$, for a destination $d \in N$, and forces that there can only be a single choice. Equation $(5.3e)$ forces commodities with destination d to follow the chosen path.

5.2.2 Example

Using example defined in Section 5.1.2, if in-tree loading is enforced and freight splitting is allowed, then the optimal solution cost is 47, with the solution given in Table 5.7 and 5.8.

Table 5.7: In-tree / Split Example Solution

Arc	**Time**	**Consolidation** (k, q)	
0,1	0	$k_1, 1.1$	$k_2, 0.8$
0,1	2	$k_1, 0.4$	$k_3, 0.6$
1,2	7	$k_1, 1.5$	

Table 5.8: In-tree / Split Temporary Commodities

k	k'	**Path**	$q^{k'}$
k_1	k_1^1	0,1,2	1.1
k_1	k_1^2	0,1,2	0.4

Continuing the analysis of the example instance, Table 5.9 shows the optimal solution cost for each of the extensions, and compares them against the original (unextended model) solution cost. Intuitively in-tree should be greater-than-equal to the original; split will be less-than-equal; and tree/split could be either.

Table 5.9: Example Cost

Algorithm	**Cost**	**Relative Cost Multiplier**
original	50	1.00
in-tree	50	1.00
split	41	0.82
tree/split	47	0.94

5.3 Computational Results

With these new extensions, there are two main questions that need to be addressed: how do they effect quality, and performance. To answer this, we would like to solve all 558 unrounded (i.e. 1 minute) CTSNDP instances from Section 2.4, to 0.001% optimality three times. First for a quality comparison, with a computational time limit of 1 day, using the Adaptive variant of DDDI; then for a performance comparison, with a computational time limit of 1 hour, using both Adaptive DDDI and Full Discretization. However, since

there are two extensions (plus one extra combined), this will require solving the instances nine times. We do not believe that much additional insight would be gained from the computational effort of solving all instances, and so we instead restrict ourselves to the HC/LF group, as this should give a representable sample (with 183 CTSNDP reasonably challenging instances).

The summarized performance results can be seen in Table 5.10. The averaged number of variables and constraints for DDDI is taken from the model of the last iteration for each instance. As expected, DDDI clearly outperforms Full Discretization (FD) for each extension. It is interesting to see that allowing freight splitting makes the problem significantly harder to solve (in both DDDI and FD), even though for FD it results in a much lower Gap %. Observe that none of the instances (when allowing freight splitting) were solved to optimality using FD. The number of constraints for (split) and (tree/split) is considerably lower than (original) or (in-tree), since the valid Inequality (4.6) is not used for these models. As shown in Section 4.7, these cuts increase the % Optimal solved within the computational limit, however, allowing freight splitting still increases the difficulty to solve (i.e. fewer instances were solved to optimality), even when this is taken into account.

Table 5.10: Extensions Performance Summary (1 hr time limit)

Algorithm	**Gap %**	**Time (s)**	**Variables**	**Constraints**	**# Iters**	**% Optimal**
FD (original)	18.28	2,595.0	676,743.6	678,349.0		32.8
FD (in-tree)	14.69	2,553.0	700,167.7	681,304.3		32.4
FD (split)	5.48	3,600.5	676,817.9	397,720.7		0.0
FD (tree/split)	5.42	3,600.6	681,481.7	390,951.5		0.0
DDDI (original)	0.12	677.8	14,013.6	25,757.0	14.8	85.8
DDDI (in-tree)	0.13	793.4	37,224.0	28,579.0	14.7	82.4
DDDI (split)	0.41	1,639.3	13,884.9	9,699.5	13.5	56.8
DDDI (tree/split)	0.56	1,651.4	36,301.6	12,076.9	12.9	57.9

The summarized quality results are shown in Table 5.11. Specifically, this shows the average relative cost of the various extensions, using the higher quality solutions (found by

solving with increased time limits). The results are somewhat surprising, as the effect is not particularly significant, especially when taking the overall Gap % into account. This

Table 5.11: Extensions Quality Summary

Algorithm	**Relative Cost Multiplier**	**Gap %**
DDDI (original)	1.0000	0.06
DDDI (in-tree)	1.0025	0.06
DDDI (split)	0.9998	0.30
DDDI (tree/split)	1.0045	0.45

is a positive result, as in-tree loading dramatically simplifies the handling and routing at terminals, and freight splitting allows for larger instances by aggregating commodities – by having costs that are relatively similar, this shows that both extensions are useful modeling approaches, and give high quality solutions.

The relative model size (from the last iteration using the 1 hour time limit) for the various extensions is shown in Figure 5.3. Note that since each extension (original, in-tree, split, tree/split) has different variables/constraints, each needs to be relative to its own full discretized model.

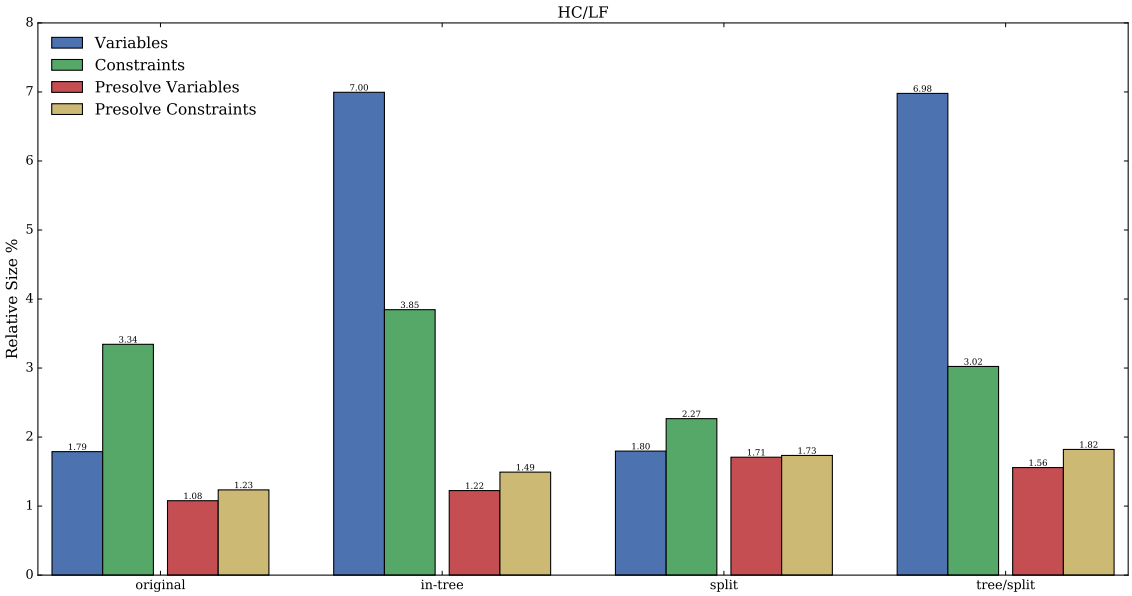


Figure 5.3: Relative Model Size (1hr solve time)

Observe that when using in-tree loading, the number of variables increases; and as mentioned earlier, the freight splitting models use fewer constraints – however after the CPLEX presolve, the number of variables and constraints are quite similar. The model growth (relative number of variables plus constraints), shown in Figure 5.4 predominantly occurs in the early iterations of DDDI.

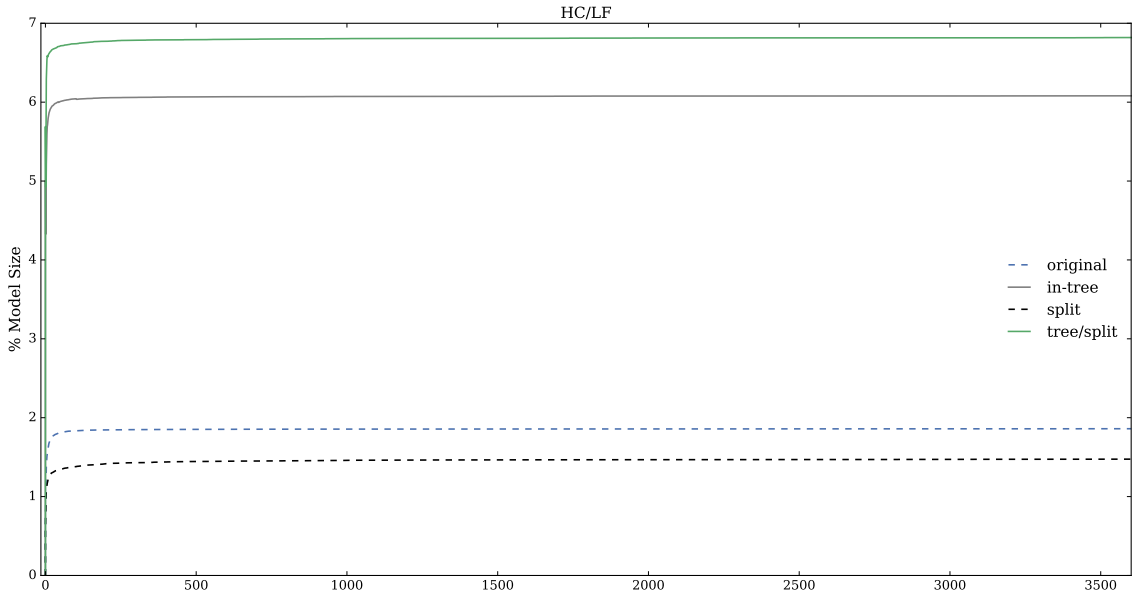


Figure 5.4: Extensions Model Growth (1hr solve time)

Since the number of presolve variables and constraints are similar, the convergence profiles in Figure 5.5 are also similar. Observe, however, that the split, and tree/split extensions take longer to converge. This is likely an effect of removing the valid Inequality (4.6), but due also to freight splitting, which may increase the symmetry of the lower bound DDDI model, that is, increased number of solutions having the same cost, but differ in the way freight is split. Note that due to the approach of DDDI, convergence should not be impacted by the symmetry of options for choosing how to split commodities over a fixed set of paths (i.e. different quantities) – but rather the increased number of options for different paths. For example, if two commodities with disjoint time-windows consolidate over an arc, then there may be multiple solutions having the same cost but with different quantities

for each of the split commodities. Regardless of all these options for choosing quantities, the unary time point in Theorem 6 will still stop these commodities from consolidating on this arc, and so it is clear that this does not impact convergence.

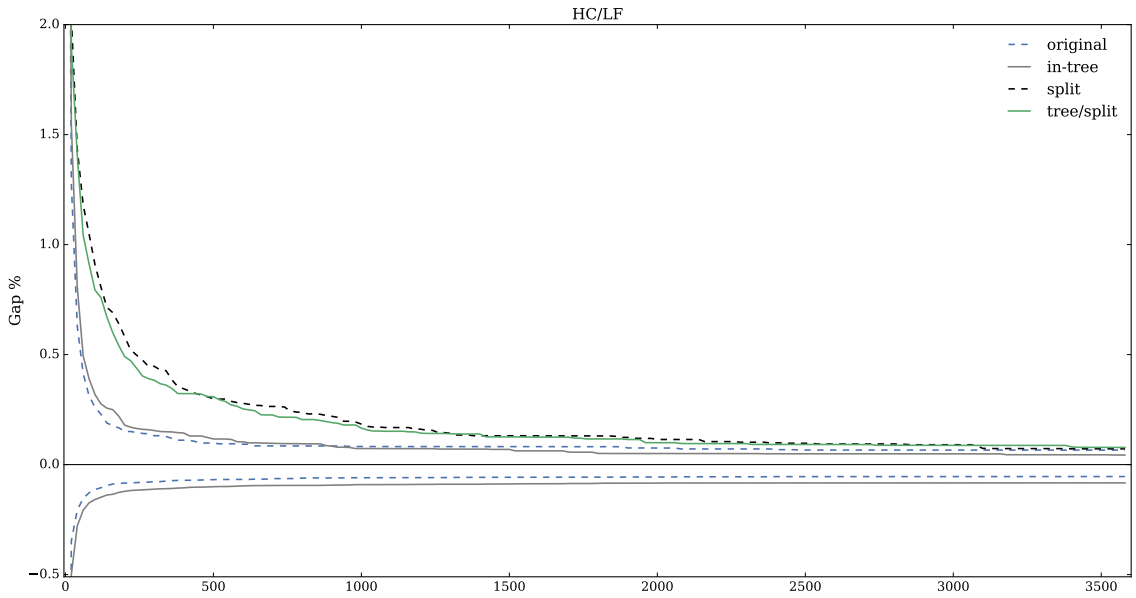


Figure 5.5: Extensions Convergence (1hr solve time)

5.3.1 Split Freight

To further investigate the splitting of freight in the high quality solutions found above (i.e. using increased time limits), see Table 5.12 for a summary. The definition of the columns are given below.

Table 5.12: Split Summary

Algorithm	**Avg #**	**Avg Max #**	**Max #**	**% Splits**	**Avg Split Size**
split	1.01	1.82	3	0.83	2.01
tree/split	1.01	1.71	3	0.55	2.01

Let $\mathcal{W}_i^k$ be the set of timed-paths for commodity $k \in K_i$ in the solution to instance $i = 1, \ldots, n = 183$. Commodity k in instance i is split when $|\mathcal{W}_i^k| > 1$. Let the average number of splits (Avg #), the average maximum number of splits (Avg Max #), and the maximum number of splits (Max #) be

$$\text{Avg \#} = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{k\in K_i}|\mathcal{W}_i^k|}{|K_i|},$$
$$\text{Avg Max \#} = \frac{1}{n}\sum_{i=1}^{n}\max_{k\in K_i}|\mathcal{W}_i^k|,$$
$$\text{Max \#} = \max_{i=1,\ldots,n,\, k\in K_i}|\mathcal{W}_i^k|.$$

To simplify notation, let $s_i^k = \mathbb{1}\{|\mathcal{W}_i^k| > 1\}$, indicate if commodity k is split for instance i; and let $\mathcal{S}_i^k = \sum_{k\in K_i} s_i^k$, be the total number of split commodities in instance i. Then, let the percentage of commodities that are split (% Splits), and the average number of splits when a commodity is split (Avg Split Size) be

$$\%\text{ Splits} = \frac{100}{n}\sum_{i=1}^{n}\frac{\mathcal{S}_i^k}{|K_i|},$$
$$\text{Avg Split Size} = \frac{1}{\sum_{i=1}^{n} s_i^k}\sum_{i=1}^{n}\begin{cases}\frac{\sum_{k\in K_i}|\mathcal{W}_i^k|s_i^k}{\mathcal{S}_i^k} & \mathcal{S}_i^k > 0\\ 0 & \text{o/w}\end{cases}.$$

As can be seen from the results in Table 5.12, very few splits occurred. Less than 1% of the commodities were split, and of these, typically the commodity is split into two pieces. The maximum number of splits is only three. Thus, freight tends to be kept together in solutions. This makes intuitive sense, since our aim is to consolidate freight, and this is the easiest way to consolidate.

Observe that when using the tree/split extension, even fewer commodities were split. This seems reasonable, since the simplified routing strategy at each terminal means that freight would only be split for different dispatch times, rather than also along different paths. Furthermore, our objective tries to consolidate freight, so the number of dispatches is kept to a minimum, which means that there are fewer reasons to split.

Part II

Service Operation

CHAPTER 6

DYNAMIC PLANNING WITH A LARGE SCALE SERVICE NETWORK

6.1 Introduction

The successful implementation of daily less-than-truckload (LTL) operations typically relies on the solutions to many related optimization problems that encompass various levels: strategic (long-term), tactical (medium-term), and operational (short-term) (Crainic and Laporte, [1997]). The Service Network Design Problem sits at the tactical level, and is used to decide how freight should flow through the network using predicted or average volumes. The resulting *planned flow* paths are often designed to make operational tasks easier in real life. For example, the in-tree loading (Section 5.2) forces shipments with common ultimate destination to be routed through the same next terminal – by doing so, this reduces the time and complexity of the *sort* (the process of unloading, allocating, and loading freight) at a terminal, and hence reduces the chances of mistakes and late delivery. On a day-to-day basis the volume of freight fluctuates from the forecasted estimates. To improve consolidation and handling of the daily freight variations, a small set of *alternate flow* paths are created, and are dynamically used at each terminal in order to balance the trade-off between efficiency and implementability.

Driver scheduling is another important tactical level problem. Typically schedules are planned a week or more in advance (ensuring enough time to coordinate staff, that is, being considerate of the quality of life for the drivers), and often needs to take into account strict labor and safety laws, as well as the expected volume, flow-paths, fleet size, and equipment balancing. These schedules are complex, and contain information for each trailer on every leg. Specifically, this includes the expected dispatch and arrival times and terminals for every leg, as well as relay information (which trailer relays to what schedule on a particular

leg). The schedule is also aware of the origin/destination for each trailer, which are the terminals where that trailer is loaded/unloaded; this information is known as a *trailer-load*. A schedule may span multiple days, or may repeat multiple times within a week; and so day-of-week information is also required. We assume that these schedules are given as input. The actual task of creating a suitable schedule is NP-hard, and there is a significant body of research covering this problem in transportation related fields.

At the operational level, actual freight volumes at each terminal are revealed every day (as opposed to the tactical level where only predicted volumes are available). This information can be used to dynamically modify the planned schedules in order to reduce costs, or to meet service level guarantees, in response to the variation between the predicted and actual volumes. This operational scheduling problem is highly dynamic and requires near immediate solutions at a large scale. This chapter investigates this problem, and proposes an algorithm for its solution - which we call the Dynamic Shipment-loading and Scheduling Heuristic (DSSH). The contributions of our work can be summarized as follows, it:

- shows an example of how to take advantage of tactical level planning (load-plans and schedules) to dynamically respond to daily changes in shipment volume at a large scale. That is, improvements can be made to load-plans and driver schedules within minutes – allowing LTL dispatchers to react efficiently. Here, a typical instance needs to route over 100,000 shipments in over 10,000 trailer-loads, originating across 700 terminals spread over the contiguous United States.

- constructs intuitive data structures and algorithms that exploit the relationships between driver schedules and trailer-loads so that schedule cancellation and modifications are easily handled.

- provides an effective urgency based scheme for shipment routing, and a fast, path based algorithm for improving on-time delivery.

- defines metrics for a comprehensive evaluation of solutions.

For the remaining of this chapter, the system described and solution approach is specific to a particular national US LTL carrier, however the concepts can be applied to many LTL carriers.

6.1.1 Less-than-Truckload System Description

A typical Less-than-Truckload (LTL) shipment occupies only 5–10% of the trailer capacity, therefore, in order to improve the trailer capacity utilization, LTL carriers collect and consolidate freight from multiple shippers, and route them together. This consolidation reduces the total fixed transportation costs to the carrier, and hence makes the service more affordable to the customers.

LTL carriers typically operate on a fixed terminal network; shipments are picked up from customers and delivered to an *end-of-line* terminal, where they are sorted and allocated to a transportation lane, and dispatched across the network to their ultimate destination. In order to increase consolidation, these in-transit shipments can be unloaded and reallocated into different trailers at *breakbulk* terminals. This type of network is known as hub-and-spoke, and is typically referred to in the LTL industry as the *line-haul* network. It is possible that a terminal is both end-of-line and breakbulk. Other "terminals" exist, for example *meet-and-turn* terminals are used by drivers as a location to swap trailers, and *rail-heads* which are used for the interchange of trailers between trucks and rail - however freight is never unloaded, nor sorted at these *relay* terminals.

Shipments arrive to a terminal (both end-of-line and breakbulk) throughout the day, but are processed in time intervals known as *sorts* (inherited from the package express business at the LTL carrier). Typically each terminal has four sorts (sunrise, day, twilight, and night), however smaller terminals may only have three (sunrise, twilight, and night), or even only two (sunrise, and twilight). The operational hours for each sort varies for each terminal and day-of-week. Each sort has a *cutoff* time; any freight that arrives to a terminal

after this time will be processed in the following sort; and a *handling* time, which is the amount of time it takes to unload/process shipments from inbound trailers. In addition to the different operational times across the different days-of-week, the terminals also span across the entire contiguous United States and thus are located in different time zones. Moreover, there exists terminals outside the United States, however these will be excluded from our consideration.

Freight is most commonly packed into 28-ft long trailers called *pups*. Typically a single driver will transport two pups at a time, however in some states three pups are allowed. Drivers can alternatively transport a single *long* (or *van*) trailer (53-ft), which is considered to be approximately equivalent, in terms of capacity, to 1.7 pups. Each shipment is considered to be indivisible, that is, it cannot be split across different trailers. Every shipment has an origin terminal, an arrival time, ultimate destination and a quantity (often measured in cube or volume, however it is also convenient to use fractional pup trailers). LTL carriers have *service level* guarantees which require shipments to be delivered to their ultimate destinations within a timely manner. This service level (measured in days) is typically common for all shipments that share the same origin, arrival sort, and ultimate destination; however individual high priority shipments may also exist.

Drivers can either be contractors, or employees. Contractors can only travel one-way (only paid for one direction of travel, and no contract should be planned in advance for a return trip), and almost always provide their own *long* trailers, whereas employees must complete a cycle, typically on the same working day, but can cross many days for sleeper teams, or two days for *turn* drivers (out one day, back the next day). Employees almost always transport two pups (owned by LTL carrier), except when transporting freight to rail-heads which predominantly use long trailers. Trailer balance is generally preserved due to the cyclical driver schedules (including those that visit a railhead), however it may be necessary to transport trailers that are empty or have low utilization. This situation is undesirable, and so schedules and *load-plans* are often designed to increase the *backhaul*

utilization.

As mentioned previously, solutions to the SNDP result in planned and alternate flow paths for the line-haul network, as well as timing and number of required trailers. The *load-plan* specifies how the LTL carrier plans to transport freight through the line-haul network, typically by defining a set of *trailer-loads*. These trailer-loads are defined by an origin, destination, and a dispatch time. Conceptually, they are physical trailers that are loaded at the origin terminal, dispatched, and then unloaded at the destination - the trailer itself may travel via multiple terminals by means of several drivers (including rail). In this way, a trailer-load may use multiple *legs* or *movements*, corresponding to the number of terminals it visits from the origin to its destination. It is assumed that the freight is not touched until it reaches its destination, however in practice this is not necessarily true - LTL carriers often *head-load* freight, which places a small number of shipments at the head of a trailer-load, to be conveniently unloaded at specific terminals along the movement path; similarly, shipments can be loaded onto a trailer-load along its movement path. Obviously this desirable, as it increases efficiency, but it does incur additional complexity and handling time (albeit reduced).

Driver scheduling is a complex and comprehensive problem, due to many real-life constraints and safety standards. However, given a set of potential schedules, a suitable plan can be determined, allocating each movement in a trailer-load to a driver schedule movement. Thus the driver schedule contains all the information about trailer-loads, and is aware of the relay information required to exchange trailers with other drivers schedules and deliver the trailer-load to its destination. Often the planned schedule is over capacitated, to easily allow for higher than average volumes of freight, however for rare circumstances there are also extra-board drivers who are on call to transport extra freight (if suitable). Conversely, if a terminal has less than expected freight, it is possible to cancel driver schedules (both contractor and employees, as long as they are notified within certain time frames), or even cancel partial schedules (as long as the employee has a return trip to their

home terminal). Employees with canceled schedules may be temporarily reassigned to sorting shipments on the dock, so that they are still paid and the LTL carrier makes efficient use of their time.

6.1.2 Related Literature

Historically, most academic attention has been given to the tactical level of transportation problems rather than at the operational level (Erera, Karacık, and Savelsbergh, [2008]), this is partly due to the infrastructure challenges of making high speed operational decisions (at a central location) spread across terminals located large distances away, and so many operational decisions were (and still are) being made at each local terminal. Over the past two decades, technological advancements have made this no longer an issue, and the research at the operational level has gained serious traction. That said, however, the research focus on tactical planning is still relevant given that operational tasks are much easier to implement (and modify) when well planned schedules are available ahead of time, and the computational challenges imposed by the network design and scheduling problems are still significantly high.

For the tactical level, Van den Bergh et al., (2013) provides an extensive literature review of the work regarding personnel scheduling problems, and Crainic and Laporte, (1997) provides the classic reference for planning in freight transportation and logistics. A recent survey on driver scheduling is given by Koubâa et al., (2016). At the operational level, Powell, (2003) gives a comprehensive overview on its real time dynamics and highlights the issues and challenges that it faces. Most research on this topic recognizes the need for solving large scaled instances, and acknowledge that exact approaches are often computationally prohibitive. Our approach wants to take advantage of the existing methods and research as much as possible. By relying heavily on tactical load-plans and planned driver schedules, much of the computational burden is lifted, allowing our algorithms to rapidly find suitable solutions.

For an example of tactical planning, Erera et al., (2012) solves a large scale load-planning problem using an IP based heuristic. They do not take schedules into account, until they extend the scope in Erera et al., (2013), which focuses more on generating schedules using a three phase approach (GRASP, LP column generation, and dynamic programming). Driving regulations for rest duration are taken into consideration when matching drivers to dispatches. Instances of comparable size to ours, are solved within 2 hours.

Our shipment-loading approach can be viewed as a recourse in a stochastic service network design problem, with unknown demand. Yang and Chen, (2009) solves a similar problem for airline freight networks; while Bai et al., (2014) investigates this for the service network, and allows rerouting as a second stage decision. This approach has many advantages for developing robust and efficient plans, however due to the additional computational complexity of stochastic programming, only instances of very small size can currently be solved.

For some LTL carriers, drivers are not pre-scheduled. Instead these operators accumulate freight at a terminal until a trailer is full (or has enough freight), then dispatch as soon as an appropriate driver can be assigned. Under this assumption, Erera, Karacık, and Savelsbergh, (2008) dynamically generates driver schedules, with rest times that follow the DOT regulations. In order to solve real life instances, they use a greedy approach with partial solution enumeration. Although pre-scheduling is becoming the common practice these days, the concepts of dynamic scheduling are still incredibly important, and algorithms such as this are considered complementary to DSSH instead of an alternative – especially in unexpected situations where the planned schedules does not cover the volume of freight appropriately.

6.2 Problem Description

The central dispatch team at the LTL carrier would benefit from an optimization support tool to assist in their daily decisions on the routing of shipments through their LTL line-haul network. This tool will work in conjunction with the many existing (but longer range) tools that help plan volume, shipment flow, loads, and schedules a week or more in advance of the day of operation. Specifically, it will help the operations team react to the daily variations in volume, and adjust the planned schedules in order to improve cost efficiency (and reduce empty miles).

We are given a line-haul system that has been designed to cost-effectively move a large volume of freight while respecting service guarantees. In addition, planned and alternate flow paths are given, such that they define a sequence of *trailer directs* (i.e. driver movements between two end-of-line or breakbulk terminals, that perhaps travel via a meet-and-turn terminal), which can be used for consolidating shipments, and designing a load-plan. Each week, to prepare for the week ahead, (driver) schedules are created. Average origin / destination volumes are used to determine the expected number of trailer-loads required between two terminals, assuming travel along the planned flow paths. The legs of a planned (driver) schedule are known as *planned movements*.

At the start of the twilight sort (around 6pm) each day the actual shipment volumes for that day are known (in truth this has approximately 80% accuracy; it is collected and registered throughout the day via the use of electronic devices known as DIADs). Since the actual volume is likely to differ from the predicted, Central Dispatch may have to adjust the driver schedules, such as organizing additional loads using extra board/external drivers, or canceling schedules, if the volume is higher or lower (respectively) than expected. In addition, adjusting a plan may involve sending freight via an alternate flow, or, more rarely, delaying freight (only at hubs) for a future dispatch.

Our aim is to assist Central Dispatch with these dynamic adjustments, using the actual

freight volume data to suggest modifications to the driver schedules, so that system wide trailer movements are more cost effective (including empty repositioning considerations). To achieve this, we will implement a tool which will be run within the twilight sort at Central Dispatch every day, around 6pm, having a computational time limit of at most 30 minutes, and will suggest a plan that is effective until 6 pm the next day.

6.2.1 Problem Data

It should be clear from the system description that many different types of data are needed to create a problem instance. At a high level, this includes: schedules, freight, and infrastructure (e.g. service network); and in addition, historical results are also needed, so that we can evaluate our solution. Specifically, the following input is assumed given (for a particular time horizon): historical trailer-loads, historical shipments, historical route of shipments, planned trailer-loads (driver schedules), real-time shipments, allowable routes (including transit time and distances), terminals, sorts, timezones, and service level guarantees (for due times).

To incorporate timezones and the day-of-week operating schedules easily, all time related information will be mapped as a relative offset to 6pm EDT, which is the time and timezone for when the tool is run. This is considered as time 0 for our algorithm. Obviously DSSH has no control over the decisions made before time 0, however these decisions impact the state of the system at (and beyond) time 0. These decisions can be categorized into *schedules* and *freight*.

6.2.1.1 Schedules

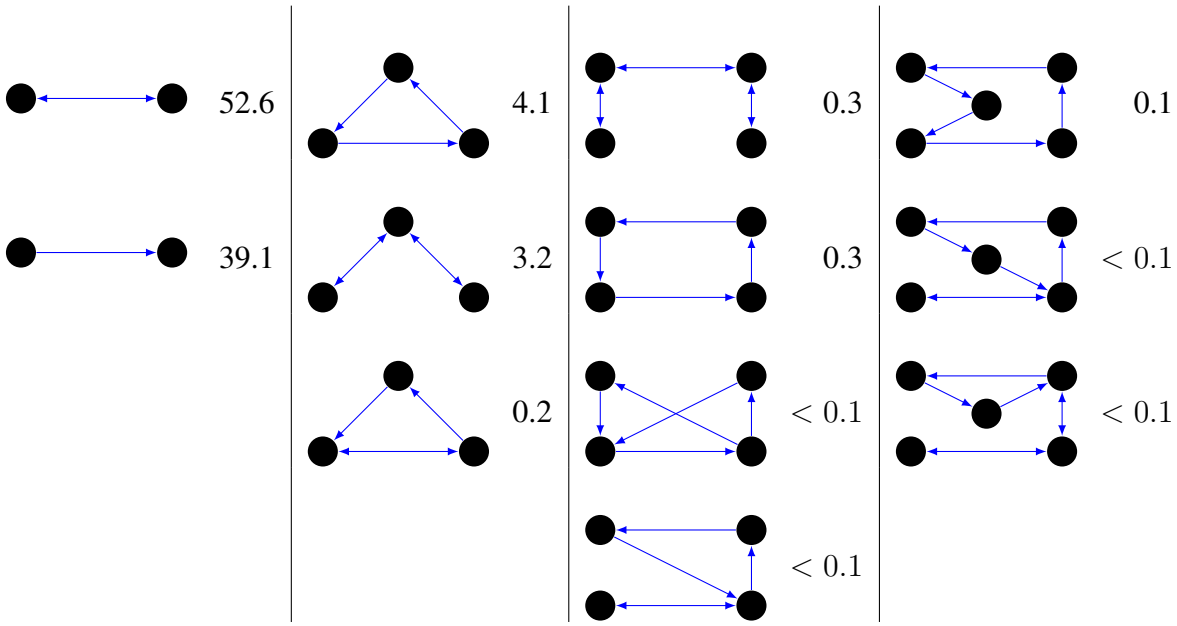
Schedules are planned as a minimum of one day of work, but often involve many legs spanning several days (up to one week). A schedule may move multiple trailers at the same time, and each trailer can have different origin/destination terminals. That is, a trailer may have been loaded at the movement origin, or relayed from another movement (with

the same or different schedule); similarly that trailer may be unloaded at the movement destination, or relayed to another movement. Recall that a movement defines a single move performed by a schedule (e.g. driver or rail) between a pair terminals.

Since many schedules span multiple days, our time-horizon will likely contain both *Active* and *Planned* schedules. Active schedules start before time 0 but have at least one incomplete movement remaining at time 0. Planned schedules have not yet started at time 0 and have at least one movement that start in the upcoming time-horizon. We assume that the active and planned schedules reflect the most recent changes to the system, however this may not actually be the case (dependent on the information systems in place at the LTL carrier).

A schedule is typically a cycle between two terminals, or, in the case of contractors and rail, is a one way between two terminals. Other more complicated schedules exist, but are far less frequent. See Table 6.1 to see the various topologies of schedules (nodes represent terminals, and arcs specify direction of travel), along with a usage percentage. Note that a single schedule may in part (or entirety) repeat multiple times.

Table 6.1: Topology Distribution of Schedules



6.2.1.2 Shipments / Freight

At time 0, freight is either associated as *inbound* or *outbound*. Inbound freight at a terminal refers to the daily picked-up shipments (available at time 0 with known quantity), as well as inbound line-haul trailers. It is assumed that the shipment information (ultimate destination, quantity, etc) is known for each shipment on the inbound trailer. Inbound freight can further be categorized as follows.

- **En route:** inbound trailers dispatched before time 0 but arriving at a terminal at or after time 0.
- **Unstripped:** inbound trailers that arrived at a terminal before time 0 but that have not yet been stripped.
- **Partially stripped:** inbound trailers that arrived at a terminal before time 0 and are being stripped at time 0.
- **Active freight on dock:** freight from inbound trailers that arrived at a terminal before time 0 and that has already been unloaded but that has not yet been assigned to/loaded onto outbound trailers.

Outbound freight refers to (line-haul) trailers that have started loading at a terminal before time 0, but are to be dispatched after time 0. Outbound trailer-loads can be either *partially loaded*, that is, more shipments can be added to the trailer before dispatch; or *closed*, which are ready for dispatch now – and no more freight may be added. Closed trailers may have been recently loaded at the terminal or might be trailers dropped off at the terminal to be relayed by another schedule (without unloading).

Note that the decisions made before time 0 to assign freight to trailers (partially loaded and closed trailers) are considered to be unchangeable, and therefore, they are considered as input to our tool.

6.3 Algorithm

The initial prototypes for DSSH involved solving various integer programming problems, however none of these were suitable given the scale and time restraints involved. Thus an iterative simulation approach was developed instead. See Algorithm 11 for high level details.

Algorithm 11 High level overview of DSSH

```
1: Preprocess loaded data
2:
3: while not terminated do
4:     Create trailer-loads from schedules
5:     Load trailers (greedily in order of dispatch time)
6:     Re-route freight in order to improve on-time delivery
7:     Adjust schedules (by canceling underutilized schedules)
8:
9: Calulate metrics
```

Due to the size and complexity of the problem, data management is an important factor to consider. Instances were stored in a Microsoft SQL Server database, and many scripts and indexes have been applied in order to load the data in a suitable format, and in a timely manner. Since not all data transformations are suitable at the database level, preprocessing within DSSH is a required first step. Here shipments outside the contiguous United States are handled (see Section 6.3.1), all time related fields (dispatch/arrival times, sort operating hours etc) are mapped to the relative time horizon, and various lookup tables are cached in memory (for performance benefits).

Although DSSH must develop a plan for the next 24 hours, using this for the planning horizon is somewhat myopic, since it ignores the flow-on consequences of its decisions. However, increasing the planning horizon raises two issues: the scale of the problem increases, and the freight quantities are unknown for subsequent days. Note that for our instances, the median service level was two days long, thus this is a reasonable choice for the planning horizon. The freight quantities used for the second day is the predicted

volume, given as input from the LTL carrier.

Once all data has been loaded and processed, the core of the algorithm is an iterative simulation. The two decisions that must be made is the routing of shipments (allocation of freight to trailer-loads), and how to modify the planned schedules. Ideally the routing of shipments should not be too complex, and the changes to the schedules should be minimal, so that actually implementing the plan is manageable.

In real life, at each terminal, shipments are loaded on to trailers, which are then dispatched with a scheduled driver, and unloaded at its destination. This process is essentially what DSSH simulates. First, trailer-loads are created from the schedules. The simulation then loads all of these trailers in order of dispatch time (keeping track of the flow of freight), which gives an initial solution. Since loading is greedy, it is possible that freight is delivered late, or not delivered at all within the time horizon – thus a post-processing step re-routes these shipments to improve on-time delivery. This might not be enough, in which case the schedule can be modified to add extra board drivers, or contractors to improve capacity. Alternatively, schedules that are underutilized might be candidates for cancellation (and any associated relayed schedules); or multi-move schedules could be replaced with shorter and simpler schedules to improve utilization. Once these changes have been made, the process is iterated with the updated schedules. DSSH terminates if there are no more improvements to be made, or if the time limit has been reached. Each of these steps will be discussed in further detail below.

6.3.1 Data Preprocessing

The scope of our problem is defined only to be within the contiguous United States, and so freight (and schedules) having origin/destination terminals outside this area need to be appropriately handled. Planned flow is used to truncate the shipments so that each starts and ends at terminals within the contiguous United States, and originate at a sort within the planning horizon. We discard any shipments not fitting within these time and spatial

bounds.

Specifically, for each shipment there is a single planned flow path to its ultimate destination. This path can be considered to be a sequence of terminal-sort pairs. To enforce our spatial bounds this sequence is truncated from the first terminal-sort that enters the contiguous United States, to the last terminal-sort contained within this region. If neither first or last exist, then the shipment is ignored; otherwise this *first* terminal-sort replaces the ultimate origin, and the terminal from this *last* terminal-sort replaces the ultimate destination. In addition to spatial truncation, the travel time along the path is considered, and the shipment is excluded if it is outside the time-horizon.

Schedules with origin/destination outside the contiguous United States are safely ignored, since the truncation of freight via planned flow eliminates the need for international trailer-loads. Furthermore, international scheduling includes flights etc, which is outside the jurisdiction of Central Dispatch.

6.3.2 Creating Trailer-loads

Schedule planning at the tactical level is typically performed using trailer-loads as input, which are then updated as the schedules are modified. Due to uncontrollable circumstances, these trailer-loads are not available as input for DSSH, however, they can be reverse-engineered from the planned schedules. This approach has some advantages, as it allows DSSH to easily modify schedules and trailer-loads simultaneously (for example, this makes it easier to generate *skip-directs*: merging two or more sequential trailer-loads into a new trailer-load in order to reduce handling costs), it also serves as a validation step for the schedules – since errors in planned schedules are easily overlooked when dealing with such a large scale.

The reverse-engineering process begins by retrieving planned schedules starting from one week in the past, until one week after our time horizon. This will help ensure that any interactions between schedules are captured, especially since schedules can be at most

one week long; this will also help determine the set of active trailer-loads at time 0. From these schedules the graph $G = (\mathcal{N}, \mathcal{A})$ is created, where nodes $\mathcal{N}$ represent a trailer-movement (leg), or an associated schedule (for lookups). Note that a schedule movement can move multiple trailers simultaneously, and each trailer corresponds to its own node. An arc connects two movements, and represents when a trailer is relayed between schedules (or to the same schedule). See Figure 6.1 for an example of two interacting schedules.

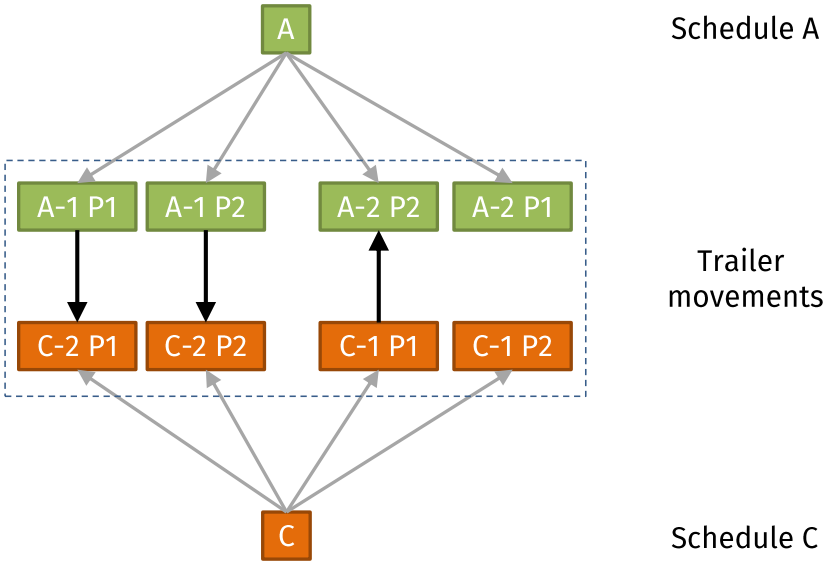


Figure 6.1: Example Schedule Graph

Although the relayed schedule is known, the appropriate leg is not, and must be determined (due to an unfortunate data issue). For example, trailer-movement (**C-1P1**), in Figure 6.1, knows that it is relayed to schedule ID (**A**), however the appropriate trailer-movement node that it relays to, must be found based on timing. This is performed by first looking at all matching schedules (i.e. there might be multiple, starting on different days), and all the associated trailer-movements which match the load origin/destination – finally then, the trailer-movement is chosen to have the earliest dispatch time after the arrival of (**C-1P1**); which happens to be (**A-2P2**).

As a consequence of this construction, by removing the schedule nodes, our reverse-engineered trailer-loads simply correspond to the paths in the connected subgraphs, as seen

in Figure 6.2

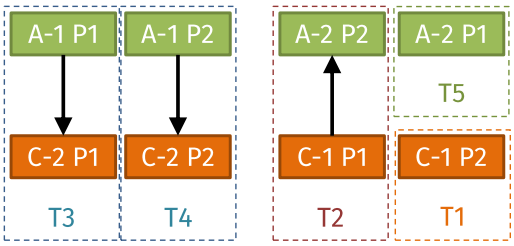


Figure 6.2: Trailer-loads as paths

6.3.3 Freight Allocation

DSSH loads one trailer at a time, processing each in the increasing order of its dispatch time, where ties are broken by decreasing miles (i.e. trailers traveling the furtherest are first). After each trailer is processed, the state of the system is updated, in particular, the transported shipments are made available at the associated destination terminal, at the arrival time + processing time. Since DSSH continues to load trailers in order of dispatch time, these shipments can again be loaded on to other trailers (dispatching from that destination terminal, with a later dispatch time), and so on, until all trailers are processed.

The loading step in DSSH assumes that these trailer-loads and dispatch times are fixed and therefore, the only decisions to be made are what shipments to load. Let $\mathcal{L}$ be the set of available trailer-loads in our time horizon, and let the functions $orig_{\mathcal{L}}(L)$, $dest_{\mathcal{L}}(L)$, $dispatch\_time_{\mathcal{L}}(L)$, $arrival\_time_{\mathcal{L}}(L)$, $capacity_{\mathcal{L}}(L)$, and $freight_{\mathcal{L}}(L)$, be defined for all $L \in \mathcal{L}$.

6.3.3.1 Allowable Routes

Let L be the current trailer-load, that is, the trailer that DSSH is about to load, and as such the current time is taken to be $dispatch\_time_{\mathcal{L}}(L)$. Let K be the set of all shipments that are available for dispatch from $orig_{\mathcal{L}}(L)$ at the current time; note that shipment processing (unloading/loading) time is already taken into account.

In order to load shipments, first consider what freight is allowed to travel in L, that is, consider if shipment $k \in K$, with origin o^k, earliest availability e^k, ultimate destination u^k, due time d^k, current terminal $orig_{\mathcal{L}}(L)$, and available quantity q^k, can travel in L. By assumption, k is available at the dispatch terminal with enough time to be loaded on to L, and for now, also assume that the quantity q^k can fit entirely in L (no splitting). With this in mind, just because k can be loaded, doesn't mean that it should be; for example, L could be traveling in the opposite direction to the ultimate destination u^k. The following is a list of routing options that could be used to dictate what freight should be loaded into L.

PF: Freight k can be loaded if $dest_{\mathcal{L}}(L)$ is the next terminal on the planned flow of k

PFX: Extending PF, freight k can also be loaded if $dest_{\mathcal{L}}(L)$ is visited anywhere along the planned flow of k (starting from $orig_{\mathcal{L}}(L)$). That is, any terminal along the path, not only the next terminal on the planned flow (i.e. PFX - planned flow extra).

PFXA: Extending PFX, freight k can also be loaded if $dest_{\mathcal{L}}(L)$ is the next terminal of any of the alternate flows of k.

PFXAX: Extending PFXA, freight k can also be loaded if $dest_{\mathcal{L}}(L)$ is visited anywhere along any planned or alternate flows. Note that this is expensive to calculate.

Head-load: Consider the movements of L. This rule allows the freight k to travel, if one of the intermediate terminals visited by L is on the planned or alternate flow of k.

The current implementation of DSSH uses **PFXA** as default, however, as described later – freight that would be traveling on its alternate flow is given special consideration. Consider now that K is the set of available shipments that also match **PFXA**. It is quite common that $\sum_{k \in K} q^k > capacity_{\mathcal{L}}(L)$, in which case, shipments should be chosen based on their priority.

6.3.3.2 Freight Priority

Maximizing the on-time delivery of freight is an important consideration. An intuitive priority scheme is to load freight in a FIFO (first-in-first-out) manner. However, this is rather simplistic. Assuming that freight k can travel on L, it can be useful to consider a list of reasons why k should not be loaded on L:

- due time violation (shipment would arrive late),
- blocking more *urgent* freight (i.e. causing other freight to arrive late),
- blocking more *constrained* freight (i.e. freight that cannot travel on many other trailer-loads),
- lack of capacity at $dest_{\mathcal{L}}(L)$ (i.e. no future trailer-loads dispatching from that terminal),
- the possibility of building a skipped direct using L, but only if k is excluded, and
- the possibility of building a skipped direct using k in a later timed trailer-load from the current terminal.

Ideally freight should be loaded into trailers such that the total amount of late freight is minimized, however for freight that arrives on-time, it is the cost of transport that should be minimized. Note that it is possible for freight to leave later, but arrive at its ultimate destination earlier. To see this, consider that an alternate flow may take a longer path; or that planned flow can have different modes of transport: rail, contractor, etc (rail typically takes longer). If, say, shipment k will be late by traveling on L, ideally future trailer-loads should then be checked to see if one allows k to arrive on time (or with reduced lateness). However, if there are a lot of these shipments, the capacity may be a limiting factor at a terminal. Thus, one possible way to load freight is by solving an LP that determines how best to allocate the known freight to the available (future) trailers at a particular terminal – however, since not all freight is known in advance, this LP would need to be solved for

each trailer-load. This approach is computationally prohibitive, and so a heuristic has been developed that attempts to address this problem.

DSSH extends the FIFO approach by first prioritizing freight based on *urgency*. The urgency of a shipment, with ultimate destination u, and due time d, currently residing at terminal n, at time t is defined as

$$U(n, t, u, d) = E(n, u, t) - (d - t),$$

where $E(n, u, t)$ is the estimated travel time (using planned flow, i.e. does not consider actual trailer-loads) from n to u if the current time is t. Thus, a positive urgency indicates freight will be late, whereas a negative number suggests freight will be on-time – furthermore large negative values correspond to increased flexibility. Ties are broken using arrival time (FIFO), then shipment quantity (largest first for best fit).

Note that the estimated travel time $E(n, t, u)$ is optimistic, because it assumes that sufficient capacity along the flow path is available. Thus, when the urgency of freight is positive, the freight is very likely to be late, that is, it is unlikely that it can be made on-time by dispatching it on any future trailer-load.

6.3.3.3 Using Alternate Flows

As hinted in the previous section, using alternate flow paths may increase utilization on a trailer-load, but it might not be a good decision for some shipments. Furthermore, it can be difficult to see all the consequences of a decision to determine if it is actually good. With this in mind, DSSH limits its use of alternate paths to shipments that would otherwise arrive late if dispatched using planned flow.

Specifically, let K_{PFX}, and K_{PFXA} be the set of shipments available and allowed to travel on L using PFX, and PFXA respectfully (and note that $K_{PFX} \subseteq K_{PFXA}$). Suppose that $\sum_{k \in K_{PFX}} q^k > capacity_{\mathcal{L}}(L)$, that is, not all shipments in K_{PFX} can be loaded into trailer L. If this is the case, shipments are loaded in order of urgency (most urgent first), then by FIFO, and finally by quantity (largest first). Shipments from $K_{PFXA} \setminus K_{PFX}$ are

not considered.

Alternatively, suppose that $\sum_{k \in K_{PFX}} q^k \leq capacity_{\mathcal{L}}(L)$, then all shipments from K_{PFX} are first loaded into L. Let shipment $k \in K_{PFXA} \setminus K_{PFX}$, and assume that k can fit in the remaining capacity. Furthermore, let $\bar{L}$ be the first trailer-load after L that is a PFX candidate for k. Clearly, if k will arrive to its ultimate destination late when traveling via $\bar{L}$, but has a chance of arriving on-time if traveling via L, then k should be loaded on L. Specifically, this occurs when

$$U(dest_{\mathcal{L}}(L), arrival\_time_{\mathcal{L}}(L), u^k, d^k) \leq 0, \text{and}$$

$$U(dest_{\mathcal{L}}(\bar{L}), arrival\_time_{\mathcal{L}}(\bar{L}), u^k, d^k) \geq 0.$$

For convenient notation, let $U(L,k) = U(dest_{\mathcal{L}}(L), arrival\_time_{\mathcal{L}}(L), u^k, d^k)$. Note that if $\bar{L}$ does not exist, i.e., the first dispatch on the flow path is after the end of the planning horizon, then only the first inequality applies, and such shipments are given the lowest priority. In this way, DSSH prefers moving freight rather than holding freight.

Suppose now that the total quantity of these valid alternate flow shipments is greater than the remaining capacity, then the shipments are loaded in order of urgency, i.e. highest values of $U(\bar{L}, k)$ first (0 if $\bar{L}$ does not exist), then by FIFO, and finally by largest quantity. See Algorithm 12 for the pseudocode.

6.3.3.4 Freight Tracking / Freight State

Since DSSH moves freight at the shipment level, the data structures required for keeping track of the system state must be efficient, while support comprehensive reporting. As such, our state is comprised of two main data structures: the *freight-event log*, and the *current-freight cache*.

A shipment is moved via a *freight-event*, which corresponds to an arrival or dispatch. These freight-events can be considered as a double-entry bookkeeping system, recording how freight is moved throughout the system during our planning horizon. The main benefit of this approach is having a full transactional log of events, which allows for the calculation

of practically any desired metric. The *current-freight cache* stores the aggregated current status (unlike *freight-event log*) of the system, and is used by the algorithm for efficient lookups. Although these data structures incur some redundancy, the benefits of its speed and flexibility are invaluable.

Algorithm 12 Allocation of Freight

for $L \in \mathcal{L}$ in order of $dispatch\_time_{\mathcal{L}}(L)$ **do**
    $loaded \leftarrow 0$
    $freight_{\mathcal{L}}(L) \leftarrow \emptyset$

    // get list of shipments, available for L using PFX scheme
    // ordered by high urgency, early time, large quantity
    for $k \in$ AVAILABLE-PFX(L) **do**
        // only dispatch up to capacity
        if $q^k + loaded \leq capacity_{\mathcal{L}}(L)$ **then**
            $freight_{\mathcal{L}}(L) \leftarrow freight_{\mathcal{L}}(L) \cup \{k\}$
            $loaded \leftarrow loaded + q^k$

        if $loaded \geq capacity_{\mathcal{L}}(L)$ **then**
            break

    // Support alternate flow paths
    // get list of shipments, available for L using PFXA scheme
    // ordered by high urgency, early time, large quantity
    for $k \in$ AVAILABLE-PFXA(L) **do**
        // would otherwise arrive late
        if $U(L,k) \leq 0$ and $U(\bar{L},k) \geq 0$ **then**
            // only dispatch up to capacity
            if $q^k + loaded \leq capacity_{\mathcal{L}}(L)$ **then**
                $freight_{\mathcal{L}}(L) \leftarrow freight_{\mathcal{L}}(L) \cup \{k\}$
                $loaded \leftarrow loaded + q^k$

        if $loaded \geq capacity_{\mathcal{L}}(L)$ **then**
            break

    // Update the state
    UPDATE-STATE(L)

6.3.4 Rerouting Late Freight

Due to the greedy loading heuristic in DSSH, it is possible that some shipments that are delivered late can actually be rerouted and be delivered on-time. Let $G_{\mathcal{L}} = (N_{\mathcal{L}}, A_{\mathcal{L}})$ be a non-uniform time-expanded network, which is constructed directly from trailer-loads. A node in $N_{\mathcal{L}}$ is a (terminal, time) pair, where time corresponds to either the dispatch or arrival time of a trailer-load. For performance and numerical stability, time (measured in hours) is rounded to two decimal places. Each arc in $A_{\mathcal{L}}$ corresponds to either a dispatch (with at least one associated trailer-load), or a holding arc.

Each dispatch arc has a list of trailer-loads, which in turn, has a list of loaded shipments and total capacity. For each shipment that is delivered late (or is late at the end of the time horizon, but not delivered), we seek to find a path through this network such that the route follows PFXA, the capacity is never violated, and that arrives at its ultimate destination on-time. The first feasible path for the shipment is chosen, by using a greedy algorithm that prioritizes earlier dispatches and fuller trailers (in order to increase utilization). Each time a suitable path is found, that shipment is rerouted by updating the system state, before processing the next shipment. Note that the shortest path is not required, only a feasible path that arrives on-time – and so with appropriate caching, this is a very efficient depth first search, see Algorithm 13 for details. Shipments without a route arriving on-time are never updated, even if a route exists that arrives earlier at its ultimate destination, but still late. The late shipments are prioritized and processed in order of due time (earliest first), and then quantity (largest first).

Algorithm 13 Reroute Freight Path

```
// finds a sequence of trailer-loads that deliver shipment k on-time
// starting node is given by (o^k, e^k)
function FIND-PATH(k ∈ K, node ∈ N_L, loads = ∅)
    // process edges with dispatch arcs first, order by earliest t_2
    // ensure that arrival is before due-time
    for ((n_1, t_1), (n_2, t_2)) ∈ A_L where (n_1, t_1) = node and t_2 ≤ d^k do
        if n_1 = n_2 then // holding arc
            return FIND-PATH(k, (n_2, t_2), loads)

        // find first trailer on this arc that matches PFXA for k, and
        // has available capacity; ordered by least remaining capacity
        if trailer ← GET-TRAILER(k, ((n_1, t_1), (n_2, t_2))) then
            if n_2 = u^k then // at destination
                return loads ∪ {trailer}
            else
                return FIND-PATH(k, (n_2, t_2), loads ∪ {trailer})
    return ∅
```

6.3.5 Manipulating Schedules

DSSH has been designed and implemented so that both freight allocation and schedule modification can easily be extended from the default schemes presented here. Many alternate strategies were experimented with, for example: prioritizing the aggregation of freight, adding optional schedules, replacing underutilized schedules, and creating skip directs. For the instances given by our LTL carrier, schedule cancellation was utmost priority, since typically the planned schedules were over-capacitated. As such, the algorithm presented here will only consider canceling schedules. Modifying and adding new schedules have been investigated, but more development is required before they can be incorporated, and additional instances are needed to verify the algorithms efficacy.

6.3.5.1 Schedule Cancellation

Once DSSH has obtained a shipment-loading plan for the entire time horizon, the solution can be further improved by canceling schedules to reduce the cost of transportation,

however, this might also cause (some) shipments to arrive late. Moreover, interconnected schedules, i.e., involving relayed trailers, must be all canceled together to avoid the risk of freight being lost or appearing at terminals unexpectedly.

Most commonly, schedules involve out-and-back trips between two terminals (one way for contractors). Other less frequently used schedules involve longer cycles (of at least three legs) – see Table 6.1 for a full breakdown. A schedule can be defined for up to one week in length, and can repeat cycles many times a day over many days. As such, being able to cancel a partial schedule can be advantageous. Thus each schedule is partitioned into its constituent cycles and each component is treated independently; although they can still be connected if trailers are relayed between the cycles. With this in mind, schedules can be considered as simple cycles (or paths for contractors and rail).

By representing the connections between schedules (i.e., trailer relays) as arcs in a graph (see Figure 6.1), the components of the graph are independent and represent cancellable units. Thus for a schedule to be a candidate for cancellation, the entire component needs to be able to be canceled. This is typically determined by calculating the utilization of all schedules in the component, and comparing to a certain threshold function.

Specifically, DSSH uses a stringent process for choosing schedules to cancel. The time horizon is first divided into 3-hour time intervals, and for each iteration in Algorithm 11, it chooses a single interval, and attempts to cancel schedules that depart only in that interval. DSSH chooses this interval using an increasing round-robin scheme. Canceling a schedule at the start of the time horizon impacts the utilization of schedules later in the time horizon. That is, there is a chance that canceling an early schedule will increase the utilization of one or more later schedules. Our interval approach attempts to account for these flow-on consequences.

A component is a candidate for cancellation if the following criteria are met: no schedules are active (starting before time 0); there exists schedules that start within the

currently selected time interval; and, the weighted utilization (freight miles / capacity miles) is below a certain threshold, given by a piecewise linear function depending on average distance of each schedule leg. By incorporating distance into the threshold function we enforce high utilization for longer trips and allow low utilization for short trips. The threshold function used by DSSH is:

$$threshold(distance, u) = \begin{cases} u \leq 0.7 & distance > 1000 \\ u \leq 0.5 & distance \in (300, 1000] \\ u \leq 0.4 & distance \in (100, 300] \\ u \leq 0.3 & \text{o/w} \end{cases},$$

where $distance$ is the average miles per schedule leg, and

$$u = \frac{\sum_L miles(L) * \sum_{k \in freight(L)} q^k}{\sum_L miles(L) * capacity(L)}.$$

Note that if a planned schedule has a trailer specifically designed to be an empty trailer-load, then the sum of freight is taken to be the capacity, i.e. $\sum_{k \in freight(L)} q^k = capacity(L)$. In this way, planned empties are more likely to be preserved.

After all candidate components are identified, the number of trailer-load dispatches that can be canceled at each terminal is calculated. Components are then canceled in order of nondecreasing utilization, but never more than 40% of the total number of dispatches at a terminal. This spreads the canceled schedules across the network, and avoids removing all capacity at a terminal.

Finally, at the very end of Algorithm 11 (outside the loop) all components (across the entire time horizon) that have utilization of zero (that is, when no freight travels on any schedule within the component) are canceled.

6.4 Metrics

Solutions to the dynamic scheduling problem are large scale, and complex. As such, it can be misleading to evaluate the quality of a solution with a single metric. This section defines a number of different metrics, which aim to highlight different key aspects of the solution; in order to truly evaluate and compare results. In what follows, each section focuses on a particular key aspect, and can be divided into metrics on the input data, and metrics on the obtained solution.

6.4.1 Available Freight

6.4.1.1 Input

Total Available Freight The total amount of freight (in fractional pups) that becomes available at some point in the time horizon to be transported over the network.

Ideal Delivered If capacity (i.e. trailer-loads) is ignored and freight is sent as early as possible using planned flow, this metric shows how much freight could be delivered in the planning horizon. This is an optimistic estimate and is useful to compare the actual quantity of delivered freight.

Freight Status At the start of the time horizon, freight can be partitioned into three categories: late, projected late, and projected on-time. Late freight is already past it's due time at time 0, projected late/on-time can be optimistically estimated by looking at the planned flow (without capacity).

6.4.1.2 Solution

Freight Status At the end of the time horizon, freight can further be partitioned by whether or not it was delivered. If it was delivered, it can either be late or on-time; while if it has not been delivered, it could be late, projected late, or projected on-time (as at the start).

Percentage PF For each non-empty trailer-load, compute the fraction of freight that is moving along the planned flow. This metric is the average over all trailer-loads (as a percentage).

6.4.2 Efficiency

6.4.2.1 Input

Capacity Miles The sum of the mileage multiplied by capacity, for all trailer-loads dispatched in the time horizon. Note that this metric is also dependent on the solution, when schedules are allowed to be canceled.

Ideal FM (Freight Miles) Similar to Ideal Delivered. By ignoring capacity of trailer-loads, and assuming that freight is sent as early as possible along planned flow, this metric first calculates the cubic mileage of freight that can travel in the planning horizon, and then divides this sum by Capacity Miles. This metric gives context to Utilization FM, that is, it gives an estimate of the upper bound of Utilization and is independent of the load-plan.

6.4.2.2 Solution

Utilization TL (Trailer-Load) Utilization describes how efficiently trailer-loads are used to transport freight. For each trailer-load, the quantity of freight is divided by the trailer capacity. The average is then taken over all trailer-loads. Note that all empty trailers are ignored, so that we can determine the typical utilization when the trailer-load is actually used (and since schedules with empty trailer-loads are potential candidates for cancellation).

Utilization FM (Freight Miles) This metric takes the cubic miles of freight moved throughout the planning horizon, divided by Capacity Miles. It has an important difference to Utilization TL, since it gives a higher weight to trailers that travel long distances – which ideally should contain as much freight as possible. Unlike Ideal FM, this result depends

on the load-plan developed by DSSH, and it must enforce trailer capacity – however, it can also consider paths that do not follow the planned flow. Here, empty trailers are included – since they are included in the Capacity Miles metric.

Realized vs Ideal To easily compare the Utilization FM and Ideal FM, this metric is simply the ratio between the two. That is, Utilization FM divided by Ideal FM.

Empty TL (Trailer-Loads) As a companion to Utilization TL, this metric calculates the percentage number of trailer-loads that are empty, by taking the total count of empty trailers divided by the total count of trailer-loads.

Empty FM (Freight Miles) Similar to Utilization FM, however empty trailers have no freight miles – so in this case the capacity multiplied by the mileage is used for each empty trailer, and the sum is then is divided by Capacity Miles.

6.4.3 Quality of Service

In order to produce a good solution there is often a trade-off between trailer-load efficiency and delivering freight on-time. In the above metrics, we have seen the quantity of freight that is late (or expected to be late), but it is also important to see the quality of service, or the lateness (in hours) of the freight.

6.4.3.1 Input

Avg Lateness of Freight Late at Start At the beginning of the planning horizon, some freight is already late. This metric calculates the cube weighted average of lateness (in hours) using only this late freight.

6.4.3.2 Solution

Avg Lateness of Delivered Freight For freight that has been delivered (at the end of the time horizon), this metric measures, in hours, the average lateness (weighted by quantity

of freight). Note that if freight is delivered on-time then it is considered to be 0 hours late.

Avg Lateness of Freight Delivered Late Similar to Avg Lateness of Delivered Freight, this calculates the cube weighted average of lateness however it only looks at freight that has been delivered late (i.e. freight that is > 0 hours late).

Avg Lateness of Freight Late at End Similar to above, however this only measures lateness on freight that has not been delivered, but is already late. Note this does not include freight that is predicted to arrive late.

6.4.3.3 Velocity

An entirely different type of metric is velocity. It can be considered as the speed that freight would need to travel in order to be delivered on-time. It uses the planned flow (and so ignores capacity and dispatch times of trailer-loads) to estimate this value. As an example, a velocity of 1.0 means that if freight could travel along the planned flow (without waiting at intermediate terminals) then, on average, all the freight would be delivered on-time. Whereas a velocity of 2.0 indicates that freight would need to, on average, travel twice as fast as the planned flow in order for the freight to be delivered on-time.

Note that these metrics do not include freight that is already late, nor freight that is already at its ultimate destination.

Velocity at Start This is the cube weighted average velocity at the start of the time horizon.

Velocity at End This is the cube weighted average velocity at the end of the time horizon.

6.5 Computational Results

DSSH is a heuristic, with no guarantees on its optimality, so in order to evaluate its effectiveness, the results must be compared to some sort of baseline. We use the historical

choices for scheduling/routing as our baseline – that is, the historical trailer-loads, and the historical routing of picked-up/en route shipments. Calculating the metrics on this historical data was quite challenging due to the inconsistencies typically found in real-life data; and much effort was spent ensuring that the effects of these errors were minimized.

The results in Table 6.2 show a comparison (averaged over 5 consecutive days) of the performance observed in the historical data and the solution produced by DSSH. The two main objectives of DSSH is to efficiently route freight, and to cancel ineffective schedules. To fully evaluate both these objectives, various configurations have been set up – where a configuration specifies different parameters to DSSH, and corresponds to a column in Table 6.2. In particular, the columns are: HISTORICAL (our baseline), which indicates how shipments were actually loaded, and what trailer-loads were actually dispatched; HTL (Historical Trailer-Loads) uses our algorithm for shipment-loading, but uses historical trailer-loads for capacity; PTL (Planned Trailer-Loads) uses our algorithm for shipment-loading, and the planned schedules for trailer-loads. Note that PTL is further divided into original (ORIG) and removed (REM), where REM indicates that DSSH has also invoked its schedule canceling routines. Since HTL uses historical trailers, these results highlight the effectiveness of our shipment-loading and routing heuristic; whereas the comparison between PTL-ORIG and PTL-REM show the effectiveness of the schedule cancellation routines. Finally the comparison between HISTORICAL and PTL-REM shows how DSSH performs overall.

A high quality solution to our problem depends on the priorities of the LTL carrier. It is typically a trade-off between many different metrics. Ideally all freight should be delivered on-time via trailers with low Capacity Miles and high Utilization FM, however this requires a balance, since having a capacity that is too low results in freight not being delivered, and a utilization that is too high often results in freight being delivered late. The aim of our heuristic therefore, is to decrease Capacity Miles while increasing the delivered on-time freight, compared to the historical baseline.

Observe that HTL increases the total delivered freight by 3.5%: on-time delivery increased by 5.4%, and late delivery decreased by 1.9%. At the same time, Utilization FM increased by 1.9%. Thus the shipment-loading heuristic has been shown to be effective.

The schedule cancellation routines decrease the Capacity Miles of planned schedules by 14.1%, and has 7.1% fewer Capacity Miles compared to the HISTORICAL. Note that removing capacity from the network directly translates to cost savings for the LTL carrier. While reducing the capacity, the same amount of freight was delivered as historical – however, an extra 3% was delivered on-time. Thus the schedule canceling routines are also shown to be effective.

6.6 Final Remarks

At the time of writing, DSSH is being evaluated at a national US LTL carrier, with promising results so far. However, there is much room for improvement. DSSH has been implemented in a generic manner that establishes the infrastructure to allow for many extensions. That is, different approaches to shipment-loading, schedule cancellation, etc are easy to incorporate. Some ideas for future development are as follows. Shipment-loading could be improved by looking ahead (instead of a purely greedy approach) at the dispatching terminal, as well as looking ahead at the destination terminal to see if skip-directs can be built. To simplify terminal operations, DSSH could also try to ensure that shipments with the same ultimate destination follow the same path (i.e. in-tree loading); and the ability to efficiently head-load would certainly be an advantage.

Schedule modifications currently only involve cancellations. Extensions can (should) be investigated in which schedules are added and changed. For example, it may be possible to replace two or more unbalanced schedules (i.e. full trailer in one direction, empty trailer on the return leg) with a single schedule – and thus reducing capacity without impacting delivery. These types of approaches have much potential, especially considering the high Empty TL/TM results in Table 6.2. This indicates that there are still a large number of

Table 6.2: Comparison of Operational Performance

	HISTORICAL	HTL	PTL-ORIG	PTL-REM
Freight				
Total available freight	4511.2	4511.2	4511.2	4511.2
Percentage PF	91.3%	86.7%	98.5%	86.1%
Ideal Delivered	66.1%	66.1%	64.5%	63.7%
Delivered	58.3%	61.9%	59.3%	58.3%
Percentage Due	41.3%	41.3%	41.3%	41.3%
Not moved in 24hrs	3.9%	1.9%	8.2%	5.9%
Freight Status at Start of Time Horizon				
Late	4.8%	4.8%	4.8%	4.8%
Proj Late	23.7%	23.7%	23.7%	23.7%
Proj On-time	71.5%	71.5%	71.5%	71.5%
Freight Status at End of Time Horizon				
Not Delivered: Late	4.6%	3.4%	4.7%	5.1%
Not Delivered: Proj Late	10.4%	10.5%	11.8%	11.3%
Not Delivered: Proj On-time	26.7%	24.2%	24.2%	25.3%
Delivered: Late	13.5%	11.6%	10.2%	10.5%
Delivered: On-time	44.8%	50.2%	49.1%	47.8%
Efficiency				
Capacity Miles	4006222	4006222	4332930	3723246
Utilization TL	53.7%	72.7%	70.1%	76.2%
Utilization FM	57.1%	59.0%	50.5%	63.1%
Ideal FM	57.2%	57.2%	56.1%	64.3%
Realized vs Ideal	100.0%	103.3%	90.3%	98.4%
Empty TL	2.0%	20.8%	32.9%	26.3%
Empty TM	1.6%	17.2%	25.3%	17.0%
Quality of Service				
Avg Lateness of Freight Late at Start	60.8	60.8	60.8	60.8
Avg Lateness of Delivered Freight	5.0	4.8	4.9	5.1
Avg Lateness of Freight Delivered Late	23.2	26.6	28.8	28.6
Avg Lateness of Freight Late at End	75.0	64.7	62.3	64.6
Velocity at Start	1.24	1.24	1.24	1.24
Velocity at End	1.41	1.36	1.10	1.31

empty trailers that cannot be simply removed.

From an algorithmic point-of-view, parallel processing could be utilized. DSSH has been designed to work on a single thread, but it is possible to run multiple instances of DSSH with different (perhaps random) parameters or alternate shipment-loading heuristics and compare the solutions for the best result. Currently DSSH runs in around 10 minutes of computational time, and so it is well within its allowed runtime, and has plenty of flexibility for increased computation.

Even without all these extensions, we have shown that DSSH is an effective tool for quickly finding improved solutions for shipment-loading and schedule cancellations. By taking advantage of the solutions to tactical level planning, it easily handles large scaled instances, and works well with the LTL carriers existing planning tools. Finally, in its development, many useful metrics were developed to comprehensively evaluate its performance, and also the performance of the LTL carrier.

REFERENCES

Ahuja, Ravindra K., Thomas L. Magnanti, and James B. Orlin (1993). *Network Flows - Theory, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice-Hall.

Andersen, Jardar, Teodor Gabriel Crainic, and Marielle Christiansen (2009). "Service network design with management and coordination of multiple fleets". In: *European Journal of Operational Research* 193.2, pp. 377–389.

Andersen, Jardar et al. (2011). "Branch and Price for Service Network Design with Asset Management Constraints". In: *Transportation Science* 45.1, pp. 33–49.

Anderson, Edward J and Peter Nash (1987). *Linear programming in infinite-dimensional spaces: theory and applications*. Wiley New York.

Anderson, EJ, P Nash, and AB Philpott (1982). "A class of continuous network flow problems". In: *Mathematics of Operations Research* 7.4, pp. 501–514.

Bai, Ruibin et al. (2014). "Stochastic service network design with rerouting". In: *Transportation Research Part B: Methodological* 60 (Supplement C), pp. 50–65.

Baumann, Nadine and Martin Skutella (2006). "Solving evacuation problems efficientlyearliest arrival flows with multiple sources". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE. IEEE, pp. 399–410.

Boland, Natashia, John Dethridge, and Irina Dumitrescu (2006). "Accelerated label setting algorithms for the elementary resource constrained shortest path problem". In: *Operations Research Letters* 34.1, pp. 58–68.

Boland, Natashia et al. (2017). "The Continuous-Time Service Network Design Problem". In: *Operations Research*.

Brumbaugh, Stephen et al. (2016). *Transportation Economic Trends*. U.S. Department of Transportation, Bureau of Transportation Statistics.

Burkard, Rainer E, Karin Dlaska, and Bettina Klinz (1993). "The quickest flow problem". In: *Zeitschrift fr Operations Research* 37.1, pp. 31–58.

Chardaire, P. et al. (2005). "Solving a Time-Space Network Formulation for the Convoy Movement Problem". In: *Operations Research* 53.2, pp. 219–230.

Crainic, Teodor et al. (2014). “Service network design with resource constraints”. In: *Transportation Science*.

Crainic, Teodor Gabriel, Antonio Frangioni, and Bernard Gendron (2001). “Bundle-based relaxation methods for multicommodity capacitated fixed charge network design”. In: *Discrete Applied Mathematics*. Combinatorial Optimization Symposium, Selected Papers 112.1, pp. 73–99.

Crainic, Teodor Gabriel and Gilbert Laporte (1997). “Planning models for freight transportation”. In: *European Journal of Operational Research* 97.3, pp. 409–438.

Crainic, T.G. (2000). “Service network design in freight transportation”. In: *European Journal of Operational Research* 122.2, pp. 272–288.

Crainic, T.G., B. Gendron, and G. Hernu (2004). “A slope scaling/lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design”. In: *Journal of Heuristics* 10, pp. 525–545.

Dash, Sanjeeb et al. (2012). “A Time Bucket Formulation for the Traveling Salesman Problem with Time Windows”. In: *INFORMS Journal on Computing* 24.1, pp. 132–147.

Dror, Moshe (1994). “Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW”. In: *Operations Research* 42.5, pp. 977–978.

Erera, Alan, Burak Karacık, and Martin Savelsbergh (2008). “A Dynamic Driver Management Scheme for Less-than-Truckload Carriers”. In: *Computers & Operations Research*. Part Special Issue: Topics in Real-time Supply Chain Management 35.11, pp. 3397–3411.

Erera, Alan et al. (2012). “Improved load plan design through integer programming based local search”. In: *Transportation Science* 47.3, pp. 412–427.

Erera, Alan L. et al. (2013). “Creating schedules and computing operating costs for LTL load plans”. In: *Computers & Operations Research*. Transport Scheduling 40.3, pp. 691–702.

Feillet, Dominique et al. (2004). “An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems”. In: *Networks* 44.3, pp. 216–229.

Fischer, Frank and Christoph Helmberg (2012). “Dynamic graph generation for the shortest path problem in time expanded networks”. In: *Mathematical Programming*, pp. 1–41.

Fleischer, Lisa and Martin Skutella (2003). “Minimum cost flows over time without intermediate storage”. In: *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, pp. 66–75.

— (2007). “Quickest flows over time”. In: *SIAM Journal on Computing* 36.6, pp. 1600–1630.

Ford, Lester Randolph and Delbert Ray Fulkerson (1958). “Constructing maximal dynamic flows from static flows”. In: *Operations research* 6.3, pp. 419–433.

— (1962). *Flows in networks*. Princeton University Press.

Gale, David (1958). *Transient flows in networks.* Tech. rep. DTIC Document.

Garcia, Renan (2009). “Resource Constrained Shortest Paths and Extension”. PhD thesis. Georgia Institute of Technology.

Ghamlouch, I., T. Crainic, and M. Gendreau (2003). “Cycle-based neighborhoods for fixed charge capacitated multicommodity network design”. In: *Operations Research* 51, pp. 655–667.

— (2004). “Path relinking, Cycle-Based Neighborhoods and Capacitated Multicommodity Network Design”. In: *Annals of Operations Research* 131, pp. 109–133.

Groß, Martin and Martin Skutella (2012). “Algorithms – ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings”. In: ed. by Leah Epstein and Paolo Ferragina. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Maximum Multicommodity Flows over Time without Intermediate Storage, pp. 539–550.

Groß, Martin et al. (2012). “Algorithms – ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings”. In: ed. by Leah Epstein and Paolo Ferragina. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Approximating Earliest Arrival Flows in Arbitrary Networks, pp. 551–562.

Hall, Alex, Steffen Hippler, and Martin Skutella (2007). “Multicommodity flows over time: Efficient algorithms and complexity”. In: *Theoretical Computer Science* 379.3, pp. 387–404.

Hane, Christopher A. et al. (1995). “The fleet assignment problem: Solving a large-scale integer program”. In: *Mathematical Programming* 70.1-3, pp. 211–232.

Hewitt, M., G.L. Nemhauser, and M.W.P. Savelsbergh (2010). “Combining Exact and Heuristic Approaches for the Capacitated Fixed-Charge Network Flow Problem”. In: *INFORMS Journal on Computing* 22.2, pp. 314–325.

Hewitt, Mike, George Nemhauser, and Martin WP Savelsbergh (2013). "Branch-and-Price Guided Search for Integer Programs with an Application to the Multicommodity Fixed-Charge Network Flow Problem". In: *INFORMS Journal on Computing* 25.2, pp. 302–316.

Hoppe, Bruce and Éva Tardos (1994). "Polynomial time algorithms for some evacuation problems". In: *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, pp. 433–441.

— (2000). "The quickest transshipment problem". In: *Mathematics of Operations Research* 25.1, pp. 36–62.

Hosseininasab, Amin (2015). "The Continuous Time Service Network Design Problem". MA thesis. University of Waterloo.

Inghels, Dirk, Wout Dullaert, and Daniele Vigo (2016). "A service network design model for multimodal municipal solid waste transport". In: *European Journal of Operational Research* 254.1, pp. 68–79.

Jarrah, Ahmad I., Ellis Johnson, and Lucas C. Neubert (2009). "Large-Scale, Less-than-Truckload Service Network Design". In: *Operations Research* 57.3, pp. 609–625.

Jarvis, John J and H Donald Ratliff (1982). "Note - Some Equivalent Objectives for Dynamic Network Flow Problems". In: *Management Science* 28.1, pp. 106–109.

Katayama, N, M Chen, and M Kubo (2009). "A capacity scaling heuristic for the multicommodity capacitated network design problem". In: *Journal of Computational and Applied Mathematics* 232.1, pp. 90–101.

Kennington, Jeffery L and Charles D Nicholson (2010). "The uncapacitated time-space fixed-charge network flow problem: an empirical investigation of procedures for arc capacity assignment". In: *INFORMS Journal on Computing* 22.2, pp. 326–337.

Kliewer, Natalia, Taïeb Mellouli, and Leena Suhl (2006). "A timespace network based exact optimization model for multi-depot bus scheduling". In: *European Journal of Operational Research* 175.3, pp. 1616–1627.

Klinz, Bettina and Gerhard J Woeginger (2004). "Minimum-cost dynamic flows: The series-parallel case". In: *Networks* 43.3, pp. 153–162.

Kobayashi, Kazuhiro and Mikio Kubo (2010). "Optimization of oil tanker schedules by decomposition, column generation, and time-space network techniques". In: *Japan Journal of Industrial and Applied Mathematics* 27.1, pp. 161–173.

Koubâa, Mayssa et al. (2016). “Truck Driver Scheduling Problem: Literature Review”. In: *IFAC-PapersOnLine*. 8 IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016 49.12, pp. 1950–1955.

Magnanti, T. L. and R. T. Wong (1984). “Network Design and Transportation Planning: Models and Algorithms”. In: *Transportation Science* 18.1, pp. 1–55.

Megiddo, Nimrod (1974). “Optimal flows in networks with multiple sources and sinks”. In: *Mathematical Programming* 7.1, pp. 97–107.

Minieka, Edward (1973). “Maximal, lexicographic, and dynamic network flows”. In: *Operations Research* 21.2, pp. 517–527.

Neumann-Saavedra, Bruno Albert et al. (2016). “Service Network Design of Bike Sharing Systems with Resource Constraints”. In: *Computational Logistics*. Springer, Cham, pp. 352–366.

Nielsen, C. A et al. (2004). “Network design formulations for scheduling U.S. Air Force channel route missions”. In: *Mathematical and Computer Modelling*. Defense transportation: Algorithms, models, and applications for the 21 century 39.6, pp. 925–943.

Powell, Warren B. (2003). “Dynamic Models of Transportation Operations”. In: *Handbooks in Operations Research and Management Science*. Ed. by S. Graves and T. A. G. de Tok. Vol. 11. Supplement C vols. Supply Chain Management: Design, Coordination and Operation. DOI: 10.1016/S0927-0507(03)11013-4. Amsterdam: Elsevier, pp. 677–756.

Powell, Warren B, Patrick Jaillet, and Amedeo Odoni (1995). “Stochastic and dynamic networks and routing”. In: *Handbooks in operations research and management science* 8, pp. 141–295.

Savelsbergh, M. W. P. (1986). “Local search for routing problems with time windows”. In: *Annals of Operations Research* 4, pp. 285–305.

Schulz, John D. (2014). *2014 State of Logistics: Less-than-truckload’s welcomed rebound*.

Skutella, Martin (2009). “An introduction to network flows over time”. In: *Research Trends in Combinatorial Optimization*. Springer, pp. 451–482.

Song, Jin-Hwa and Kevin C. Furman (2013). “A maritime inventory routing problem: Practical approach”. In: *Computers & Operations Research*. Transport Scheduling 40.3, pp. 657–665.

Teypaz, Nicolas, Susann Schrenk, and Van-Dat Cung (2010). “A decomposition scheme for large-scale Service Network Design with asset management”. In: *Transportation Research Part E: Logistics and Transportation Review* 46.1, pp. 156–170.

Tjandra, Stevanus Adrianto (2003). “Dynamic network optimization with application to the evacuation problem”. PhD thesis. Universitat Kaiserslautern.

Topaloglu, Huseyin and Warren B Powell (2006). “Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems”. In: *INFORMS Journal on Computing* 18.1, pp. 31–42.

Vaidyanathan, B., K.C. Jha, and R.K. Ahuja (2007). “Multicommodity network flow approach to the railroad crew-scheduling problem”. In: *IBM Journal of Research and Development* 51.3.4, pp. 325–344.

Van den Bergh, Jorne et al. (2013). “Personnel scheduling: A literature review”. In: *European Journal of Operational Research* 226.3, pp. 367–385.

Wang, Xiubin and Amelia C. Regan (2002). “Local truckload pickup and delivery with hard time window constraints”. In: *Transportation Research Part B: Methodological* 36.2, pp. 97–112.

— (2009). “On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints”. In: *Computers & Industrial Engineering* 56.1, pp. 161–164.

Wieberneit, Nicole (2008). “Service network design for freight transportation: a review”. In: *OR Spectrum* 30.1, pp. 77–112.

Yaghini, Masoud, Mohadeseh Rahbar, and Mohammad Karimi (2012). “A hybrid simulated annealing and column generation approach for capacitated multicommodity network design”. In: *Journal of the Operational Research Society* 64.7, pp. 1010–1020.

Yan, Shangyao and Yu-Lin Shih (2007). “A time-space network model for work team scheduling after a major disaster”. In: *Journal of the Chinese Institute of Engineers* 30.1, pp. 63–75.

Yang, T. H. and C. C. Chen (2009). “Air freight service network design for stochastic demand”. In: *2009 IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics*. 2009 IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics, pp. 261–265.

VITA

Luke Marshall was born in a small town within New South Wales, Australia. He first started his career in software development, but in 2007, he decided to pursue his love of mathematics, and by 2011 he had completed his B.Sc. at the University of Queensland with a major in Statistics, and an extended major in Mathematics. In 2012, he continued his study and achieved a B.Sc. Honours (Class I) in Mathematics, receiving the University Medal in recognition of his outstanding academic merit. Throughout the duration of his studies, he continued to work almost full time as a software developer, working on various applications involving spectral analysis using electron microscopes, printing infrastructure, and spatial data analysis. After traveling around the world in 2013, he joined the PhD program, in 2014, studying Operations Research (with a minor in machine learning) at the H. Milton Stewart School of Industrial and Systems Engineering at the Georgia Institute of Technology. His advisors Drs. Natashia Boland, and Martin Savelsbergh, guided his research in time-expanded networks and applications to logistics. He also collaborated with Drs. Mike Hewitt, Alan Erera, and Iman Dayarian on various projects relating to his dissertation.

Luke is interested in many aspects of mathematical optimization (both applied and theoretical), and loves to combine this with software development. After graduation he plans to start working as a Researcher at Microsoft Research, Redmond, Washington, in the spring of 2018.